НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.912

«До захисту допущено»
Завідувач кафедри
_____ І.А. Дичка
«___»_____2019 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Метод предиктивного введення тексту на мобільних пристроях»

Виконав:
студент ІІ курсу, групи КП-71мн
Копач Сергій Миколайович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Тетяна Миколаївна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н.,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри ММСА, к.т.н., доцент,
Дідковська Марина Віталіївна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський) за освітньо-науковою програмою

Спеціальність (спеціалізація) – 121 «Інженерія програмного забезпечення» («Програмне забезпечення комп'ютерних та інформаційно-пошукових систем»)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ І.А. Дичка

«___»_____2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Копачу Сергію Миколайовичу

1. Тема дисертації «Метод предиктивного введення тексту на мобільних пристроях», науковий керівник дисертації Заболотня Тетяна Миколаївна, к.т.н., доцент, затверджені наказом по університету від «8»квітня 2019 р. №1075-С

2. Термін подання студентом дисертації «17» травня 2019 р.

3. Об'єкт дослідження: процес введення текстових даних на сучасних мобільних пристроях.

4. Предмет дослідження: методи, способи та алгоритми предиктивного введення тексту на сучасних мобільних пристроях.

5. Перелік завдань, які потрібно розробити:
- провести аналіз існуючих методів предиктивного введення тексту на мобільних пристроях та програмних засобів, що реалізують ці методи;
- дослідити ефективність засобів введення тексту на мобільних пристроях;
- запропонувати модифікацію методу предиктивного введення тексту на мобільних пристроях;
- розробити програмне забезпечення для реалізації та тестування запропонованої модифікації методу предиктивного введення тексту на мобільних пристроях;
- провести аналіз отриманих результатів.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
- схема процесу створення *N*-грам;
- схема розподілу літер між блоками таким чином, що кожній літері відповідає один блок;
- схема розподілу літер між блоками таким чином, що кожна літера утворює свій окремий блок;
- схема роботи запропонованої модифікації методу;
- архітектура розроблених програмних засобів;
- результати проведеного тестування.

7. Орієнтовний перелік публікацій:

- Тези доповіді “Модифікований метод предиктивного введення тексту на мобільних пристроях на основі *N*-грам”
- Стаття “Особливості програмної реалізації модифікованого методу предиктивного введення тексту на мобільних пристроях на основі *N*-грам”

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент кафедри ПЗКС		

9. Дата видачі завдання «11» жовтня 2017 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	17.12.2017	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	04.03.2018	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	16.05.2018	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення; підготовка матеріалів доповіді на конференції ПМК-2018	14.10.2018	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	15.12.2018	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	20.02.2019	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу	16.04.2019	
8.	Оформлення текстової і графічної частини магістерської дисертації	06.05.2019	

Студент С.М. Копач

Науковий керівник дисертації Т.М. Заболотня

РЕФЕРАТ

Актуальність теми. Обмін текстовими повідомленнями користувачами мобільних пристроїв сьогодні набув широкого розповсюдження. З огляду на це, а також на обмеженість розмірів екранної клавіатури, ефективність засобів введення тексту на мобільних пристроях має велике значення.

Існує багато методів введення тексту з клавіатури, які дозволяють у тій чи іншій мірі прискорити цей процес. Однак більшість з них не можуть бути застосовані до сучасних мобільних пристроїв, оскільки були розроблені ще за часів використання 12-кнопкових клавіатур. Новітні методи також не є оптимальними, тому що не враховують можливості помилкового натискання сусідніх літер.

Таким чином, підвищення ефективності введення тексту на сучасних мобільних пристроях шляхом розроблення нового чи модифікації існуючих методів предиктивного введення текстових даних є актуальною задачею.

Об’єктом дослідження є процес введення текстових даних на сучасних мобільних пристроях.

Предметом дослідження є методи, способи та алгоритми предиктивного введення тексту на сучасних мобільних пристроях.

Мета роботи: підвищення швидкості введення тексту на сучасних мобільних пристроях в умовах обмежених розмірів екранної клавіатури за рахунок розроблення та реалізації модифікованого методу предиктивного введення текстових даних на мобільних пристроях та відповідних програмних засобів.

Методи дослідження. В роботі використовуються методи комп’ютерної лінгвістики, статистичні методи та емпіричні методи.

Наукова новизна роботи полягає у наступному:

1. Запропоновано модифікований метод предиктивного введення текстових даних на мобільних пристроях на основі *N*-грам, який відрізняється від існуючих методів застосуванням розподілу літер між блоками таким чином, що кожна літера утворює свій окремий блок, і у такий спосіб сприяє додаванню у список пропозицій необхідних слів навіть при помилкових натисканнях сусідніх літер під час введення тексту, що сприяє прискоренню процесу введення тексту.

Практична цінність отриманих в роботі результатів полягає в тому, що використання запропонованої модифікації методу предиктивного введення текстових даних на основі *N*-грам дає можливість збільшити швидкість введення тексту в умовах обмежених розмірів екранних клавіатур сучасних мобільних пристроїв за рахунок того, що враховуються можливі помилкові натискання сусідніх літер при введенні деякого слова, що є достатньо ймовірним під час швидкого введення тексту.

Також в рамках даного дослідження розроблено програмне забезпечення для реалізації та тестування запропонованої модифікації методу предиктивного введення текстових даних, яке може бути використане при подальшій роботі над цією тематикою.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на XI науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2018-2 (Київ, 14-16 листопада 2018 р.).

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів, висновків та додатків.

У вступі надано загальну характеристику роботи, виконано оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень.

У першому розділі охарактеризовано задачу введення текстових даних на мобільних пристроях, проаналізовано існуючі методи введення тексту на мобільних пристроях та програмні засоби, що реалізують ці методи, а також розглянуто проблему недостатньої ефективності існуючих засобів предиктивного введення тексту на мобільних пристроях.

У другому розділі проаналізовано критерії ефективності методів введення тексту на мобільних пристроях, обґрунтовано вибір методу для модифікації, а також запропоновано модифікацію методу предиктивного введення тексту на мобільних пристроях на основі *N*-грам.

У третьому розділі сформульовано вимоги до розроблюваних програмних засобів для реалізації і тестування модифікованого методу предиктивного введення тексту на мобільних пристроях на основі *N*-грам, обґрунтовано вибір мови програмування, наведено структурну організацію розроблених програмних засобів, а також наведено опис структур даних, що використовуються для зберігання даних.

У четвертому розділі наведено характеристику тестових наборів даних, здійснено порівняння отриманих в ході тестування розроблених програмних засобів результатів з використанням різних наборів вхідних даних, а також запропоновано можливі шляхи подальшого вдосконалення.

У висновках проаналізовано отримані результати роботи.

У додатках наведено копію слайдів презентації та текст розроблених програмних засобів.

Робота виконана на 106 аркушах, містить 2 додатки та посилання на список використаних літературних джерел з 29 найменувань. У роботі наведено 31 рисунок та 9 таблиць.

Ключові слова: метод предиктивного введення тексту, *N*-грама, уніграма, біграма, триграма, мобільні пристрої.

ABSTRACT

Theme urgency. The users of mobile devices widely use text messages for communication today. Considering the popularity of text messages and the small size of the modern screen keyboards, the efficiency of text input methods on mobile devices is important.

A lot of text input methods that allow to increase the speed of entering the text using the keyboard exist. However, the majority of them can not be applied to the modern mobile devices, because they were developed in the era of wide usage of the 12-button keyboards. The modern text input methods are not optimal as well, because they do not take into account the possibility of inadvertent tapping the adjacent letter.

Therefore, increasing the efficiency of the text input on modern mobile devices by means of developing a new one or modifying existing predictive text input methods is an important task.

Object of research is a process of text input on modern mobile devices.

Subject of research is a set of predictive text input methods and algorithms on modern mobile devices.

Research objective: increasing the speed of the text input on modern mobile devices in conditions of small size of the screen keyboard by means of development and implementation of the modified predictive text input method on mobile devices and the related software.

Research methods. Methods of computational linguistics, statistical methods and empirical methods are used in the research.

Scientific novelty consists in the following:

1. A modified predictive text input method on mobile devices based on the *N*-grams is proposed. The modified method differs from existing methods by using the distribution of the letters among the blocks in such a way that every letter creates its own block, which lets the necessary

words to be included to the proposition list even when the adjacent letters are inadvertently tapped. This, in turn, contributes to the increase of the speed of the text input process.

Practical value of the obtained results consists in the following: the utilization of the proposed modification of the predictive text input method based on the *N*-grams allows to increase the speed of the text input in conditions of small size of the screen keyboards on modern mobile devices. This can be achieved due to taking into account the possible inadvertent tapping the adjacent letter, which can be likely enough when quickly entering text.

Also, the software was developed during this research in order to implement and test the proposed predictive text input method modification. The software can be also used for further research on this subject.

Approbation. The basic points and outcomes of the research have been presented and discussed at the 11th scientific conference for students and postgraduates «Applied mathematics and computing» PMK-2018-2 (Kyiv, November 14-16, 2018).

Structure and content of the thesis. The master thesis consists of the introduction, four chapters, conclusions and appendixes.

The introduction presents the general description of the research, gives the overview on a current state of the scientific problem, explains the research topicality, formulates the objective and tasks of the research.

In the first chapter the task of text input on mobile devices is described; existing text input methods on mobile devices and software that implements these methods are discussed; the problem of low efficiency of the existing predictive text input methods on mobile devices is discussed.

In the second chapter the criteria of efficiency of text input methods on mobile devices are analyzed; the choice of a method for modification is made and substantiated; the modification of the predictive text input method on mobile devices based on the *N*-grams is proposed.

In the third chapter the requirements for the software that is being developed to implement and test the modified predictive text input method on mobile devices based on the *N*-grams are formulated; the choice of the programming language is made and substantiated; the structural organization of the developed software is described; the data structures that are used for storing data are described.

In the fourth chapter the description of test data sets is described; the results that were received during the testing of the developed software using different data sets are analyzed; the possible ways for further research are proposed.

In the conclusions the general conclusions on the presented thesis are given; the obtained results are analyzed.

In the appendixes the copy of the slides of the presentation is presented; the source code of the developed software is presented.

The thesis is presented in 106 pages, it contains 2 appendixes and 29 references to the used information sources. 31 figures and 9 tables are given in the thesis.

Key words: predictive text input method, *N*-gram, unigram, bigram, trigram, mobile devices.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

API – Application Programming Interface (прикладний програмний інтерфейс);

ARM – Advanced RISC Machine (вдосконалена RISC-машина);

CLR – Common Language Runtime (загальномовне виконуюче середовище);

CPM – Characters per Minute (символів на хвилину);

CPS – Characters per Second (символів на секунду);

ETL – Extract, Transform, Load (витяг, перетворення, завантаження);

GPL – General Public License (загальна публічна ліцензія);

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту);

IDL – Interface Description Language (мова опису інтерфейсів);

JIT-компіляція – Just-in-time compilation (компіляція на льоту);

JVM – Java Virtual Machine (віртуальна машина Java);

KPSC – Keystrokes per Character (кількість натискань, необхідних для введення одного символу);

LLVM – Low Level Virtual Machine (низькорівнева віртуальна машина);

MLE – Maximum likelihood estimation (метод максимальної правдоподібності);

SMS – Short Message Service (служба коротких повідомлень);

ОС – операційна система.

ВСТУП

Одним з основних напрямів використання мобільних пристроїв сучасним користувачем є обмін інформацією з іншими користувачами в рамках інформаційних мереж, що використовують різні канали зв'язку – SMS, соціальні мережі, голосовий зв'язок та інші. Однак більшість з них базуються на текстовому введенні даних. Середньостатистичний користувач сучасних мобільних пристроїв регулярно використовує сервіси, що надають послуги з обміну текстовими повідомленнями – iMessage, Viber, WhatsApp, Telegram та соціальні мережі, в яких обмін текстовими повідомленнями є також однією з ключових можливостей – VK, Facebook, Instagram, Twitter тощо.

Враховуючи активність використання вищезазначених сервісів, слід зазначити, що користувачі витрачають для цього достатньо велику кількість часу, оскільки доступність цих сервісів сьогодні є надзвичайно високою – обмін текстовими повідомленнями за допомогою SMS є можливим більш ніж на 98% території України (це стосується також і інших країн, однак для прикладу взято Україну), а за допомогою сервісів, що працюють на основі Інтернету – більше 95%. Крім того, протягом останніх років швидкими темпами відбувається розповсюдження і запровадження високошвидкісного Інтернету на базі технологій зв'язку третього та четвертого покоління, що ще більше сприяє активному використанню користувачами мобільних пристроїв сервісів обміну даними.

Отже, оскільки середньостатистичний користувач проводить багато часу, відправляючи та отримуючи текстові повідомлення, оптимізація роботи користувача із текстом та скорочення часу набору тексту є важливим завданням.

Існує багато методів введення тексту з клавіатури, які дозволяють у тій чи іншій мірі прискорити цей процес. Однак більшість з них не можуть

бути застосовані до сучасних мобільних пристроїв, оскільки були розроблені ще за часів використання 12-кнопкових клавіатур. Новітні методи також не є оптимальними, тому що не враховують можливості помилкового натискання сусідніх літер або ж не забезпечують можливості пристосування до індивідуальних особливостей введення тексту тим чи іншим користувачем, таких як використання специфічної лексики чи скорочень.

Таким чином, підвищення ефективності введення тексту на сучасних мобільних пристроях шляхом розроблення нового чи модифікації існуючих методів предиктивного введення текстових даних є актуальною задачею.

1. АНАЛІЗ ПРОБЛЕМИ І ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1. Характеристика задачі введення текстових даних на мобільних пристроях

Сьогодні у світі все більше людей використовують у своєму повсякденному житті мобільні пристрої. Так, вже в 2016 році обсяги використання мобільних пристроїв – телефонів, смартфонів, планшетів тощо для доступу до Інтернету перевищили обсяги використання настільних комп'ютерів та ноутбуків. Загальний графік зміни співвідношення використання мобільних та настільних пристроїв для доступу до Інтернету у світі за останні 10 років наведено на рис. 1 [1].

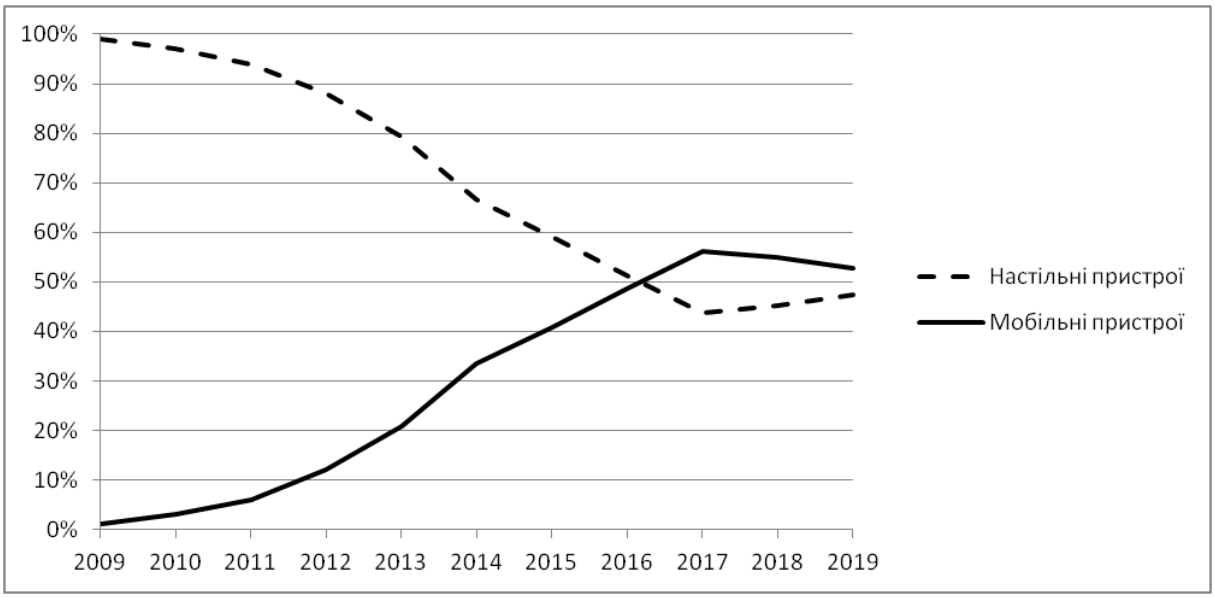


Рис. 1. Співвідношення використання настільних та мобільних пристроїв для доступу до Інтернету у світі протягом 2009-2019 років

Відповідно, зростають обсяги використання мобільних пристроїв для обміну текстовими повідомленнями та для введення текстових даних загалом. Враховуючи те, що розміри клавіатур на мобільних пристроях є суттєво меншими за розміри клавіатур на настільних комп'ютерах та ноутбуках, ускладнюється процес введення тексту, і, відповідно,

збільшується час перебування користувача перед екраном мобільного пристрою, необхідний для введення тексту.

Окрім більш тривалого перебування самого користувача перед екраном мобільного пристрою, слід також зазначити і про більш високу швидкість розрядження акумуляторної батареї гаджета, оскільки, як відомо, робота дисплею мобільного пристрою є однією з найбільш витратних з точки зору ресурсів акумуляторної батареї. Так, у порівнянні із загальним рівнем споживання ресурсів акумуляторної батареї мобільним пристроєм, при активному використанні пристрою на роботу дисплею може витрачатися близько 30% заряду акумуляторної батареї [2].

Ще одним компонентом, який впливає на роботу користувачів із мобільним пристроєм, є те, що наявність неправильних підказок засобів предиктивного введення тексту відволікає користувачів від безпосередньо процесу введення тексту та заважає швидкому викладенню думок, здійснюючи при цьому негативний емоційний вплив на людину.

Крім вищезазначених складових даної проблеми, існує також і підвищення ймовірності появи помилок при написанні тексту, що, в свою чергу, хоч і незначною мірою, може сприяти зниженню загального рівня грамотності користувачів мобільних пристроїв – а відтак, і населення загалом.

1.2. Огляд існуючих методів введення тексту на мобільних пристроях

Більшість існуючих методів введення тексту були розроблені ще за часів широкого використання мобільних пристроїв із 12-кнопковою клавіатурою. Однак, основні принципи роботи багатьох з цих методів можуть бути застосовними в тому числі й для сучасних мобільних пристроїв із сенсорними екранами, введення тексту на яких здійснюється за допомогою віртуальної екранної клавіатури. Така клавіатура, на відміну від клавіатур, що встановлювались на більшості мобільних пристроїв 2000-

х років, містить не 12 кнопок, кожна з яких відповідає певному набору літер або символів, а окремі кнопки для кожної або більшості літер.

Іншими словами, спосіб введення тексту на сучасних мобільних пристроях є схожим із способом введення тексту на звичайних клавіатурах, що зазвичай використовуються для введення тексту на настільних комп'ютерах та ноутбуках.

1.2.1. Поняття багатозначності та однозначності методів введення тексту на мобільних пристроях

Під час розгляду існуючих методів введення тексту на мобільних пристроях використовуватимемо поняття багатозначності та однозначності методів введення тексту. Так, багатозначним (англ. «ambiguous» – багатозначний, неоднозначний, невизначений) методом введення тексту вважатимемо метод введення тексту, в якому послідовність введених символів, або клавіш, що були натиснуті під час введення тексту, може відповідати одночасно декільком словам у словнику. Відповідно, у якості однозначного (англ. «unambiguous» – однозначний, недвозначний) методу введення тексту розглядатимемо метод введення тексту, в якому послідовність введених символів, або клавіш, що були натиснуті під час введення тексту, може одночасно відповідати лише одному слову у словнику.

1.2.2. Класифікація методів введення тексту на мобільних пристроях

Існуючі методи введення тексту на мобільних пристроях можна умовно розділити на наступні групи:

1) методи багатьох натискань;
2) методи одного натискання;
3) методи предиктивного введення тексту на основі *N*-грам.

Далі розглянемо ці групи методів більш детально.

Методи багатьох натискань

Методи багатьох натискань (англ. «Multi-Press Methods») – методи введення тексту, згідно з якими введення тексту забезпечується шляхом натискання більше однієї клавіші (англ. «keystroke» – натискання клавіші, натискання кнопки) для введення одного символу (англ. «character» – символ, знак) [3]. Такі методи дозволяють здійснювати однозначне введення тексту, оскільки користувач так само, як і під час введення тексту за допомогою клавіатури настільного комп'ютера або ноутбука, явно вказує символ, який необхідно відобразити на екрані. Методи багатьох натискань можуть застосовуватися як окремо, так і в складі систем, що використовують більш складні та комплексні методи введення тексту. Також, методи багатьох натискань є пристосованими для введення слів, що відсутні у словнику, оскільки явний вибір символів для введення, який має місце у даному випадку, не вимагає використання словника для забезпечення введення тексту.

Метод Multi-Tap

Метод Multi-Tap, або метод багатьох натискань (англ. «The Multi-Tap Method») був першим і найбільш поширеним методом введення тексту на мобільних пристроях, що комплектуються 12-кнопковою клавіатурою [3]. Так, згідно даного методу, для введення кожної з літер, що розміщуються на клавіатурі поряд із позначенням цифр від 2 до 9, необхідним є натискання клавіші із позначенням відповідної цифри таку кількість разів, яка відповідає порядковому номеру цієї літери на відповідній клавіші.

Наприклад, оскільки літери латинського алфавіту "a", "b" та "c" розташовуються на одній і тій самій клавіші – із позначенням цифри "2", то для введення літери "a" необхідним є одне натискання клавіші із цифрою "2", для введення літери "b" – два натискання цієї клавіші, для введення літери "c" – три натискання. Наприклад, введення слова "dog" здійснюється за допомогою послідовних натискань клавіш із позначеннями цифр "3", "6", "6", "6", "4", де одне натискання клавіші із цифрою "3"

забезпечує введення літери “d”, три натискання клавіші із цифрою “6” забезпечують введення літери “o”, а одне натискання клавіші із цифрою “4” забезпечує введення літери “g”.

Слід також зазначити, що, в залежності від конкретної реалізації даного методу, можуть відбуватися різні дії при послідовному натисканні клавіші із позначенням деякої цифри більшу кількість разів, аніж кількість літер, що розміщені на цій клавіші. Так, наприклад, при натисканні цієї клавіші таку кількість разів, що перевищує кількість літер, які розміщені на ній, на одиницю, може відбуватися введення цифри, що позначена на цій клавіші. Наприклад, чотири послідовні натискання клавіші із цифрою “2” можуть забезпечувати введення цифри “2”, чотири послідовні натискання клавіші із цифрою “3” – введення цифри “3” і так далі.

При послідовному натисканні ж клавіші із позначенням деякої цифри таку кількість разів, що перевищує кількість літер, які розміщені на ній, на два, може, наприклад, відбуватися повернення до першої літери, розміщеної на цій клавіші, в разі послідовного натискання цієї клавіші кількість разів, що перевищує кількість літер, розміщених на ній, на три – до другої літери і так далі.

Іншим прикладом є забезпечення можливості введення символів із діакритичними знаками (англ. «diacritic», «diacritical mark», «diacritical point», «diacritical sign», «accent») – надрядковими, підрядковими або внутрішньорядковими знаками при літері, що вказують на вимову, яка відрізняється від вимови звука, позначеного цією ж літерою без знака. Прикладами символів із діакритичними знаками можуть бути символи “é”, “è” та “ê”, що змінюють або уточнюють значення букви “e”, або символи “á”, “à” та “â”, що, в свою чергу, змінюють або уточнюють значення букви “a”. Деякі літери українського алфавіту також мають діакритичні знаки, наприклад такі літери як “і”, “ї” або “й”. Застосування такого підходу також може бути доцільним, оскільки введення символів із діакритичними знаками за допомогою, наприклад, використання деякого додаткового

меню, в якому містяться символи із діакритичними знаками, може потребувати відносно великої кількості часу.

Також, слід зазначити, що при послідовному натисканні клавіші із позначенням деякої цифри таку кількість разів, що перевищує кількість літер, які розміщені на ній, може відбуватися застосування одночасно обох підходів, зазначених вище. Так, наприклад, спочатку користувачеві можуть бути запропоновані до введення символи із діакритичними знаками, а при наступному послідовному натисканні цієї ж клавіші може відбуватися перехід до першої літери, розміщеної на цій клавіші.

Ще одним важливим фактом, який слід брати до уваги, є те, що при введенні деяких слів може виникнути ситуація, коли дві послідовні літери у слові розташовані на одній і тій самій клавіші. Наприклад, у слові “no” обидві літери “n” та “o” розміщуються на клавіші із позначенням цифри “6”, а для введення всіх чотирьох літер слова “feed” необхідним є натискання однієї і тієї самої клавіші – клавіші із позначенням цифри “3”. Враховуючи те, що послідовні повторні натискання однієї і тієї самої клавіші забезпечують перехід до наступної літери, розміщеної на цій клавіші, необхідним є встановлення деякого часу очікування (англ. «timeout»), після якого відбуватиметься автоматичне введення літери, що на цей момент пропонується користувачеві для введення. Ще одним з можливих варіантів вирішення даної проблеми може бути здійснення користувачем перед введенням наступної літери, що розташована на тій самій клавіші, що й попередня, натискання іншої клавіші, а після цього – клавіші стирання введеного символу, або ж натискання клавіші із позначенням стрілки, напрямленої праворуч.

Метод перевизначеної клавіатури

Оскільки на одній кнопці 12-кнопкової клавіатури міститься декілька літер, то введення кожної з них потребує різної кількості натискань. На більшості мобільних пристроїв, що комплектуються 12-кнопоковою клавіатурою, розташування літер на клавішах відбувається у алфавітному

порядку. Таке розташування є легким і зрозумілим для сприйняття користувачем, однак не є оптимальним з точки зору швидкості введення тексту. Це пов’язано з тим, що, наприклад, найбільш часто вживаною літерою англійського алфавіту є літера “e”, однак її введення потребує двох натискань.

Враховуючи це, було розроблено метод перевизначеної клавіатури (англ. «Remapped Keyboard»), основне завдання якого полягало у розміщенні в якості перших літер кожної з клавіш літер, які є найбільш вживаними, що, в свою чергу, забезпечує можливість введення таких літер за допомогою здійснення лише одного натискання.

Метод перевизначеної клавіатури використовується у програмі MessagEase, представленій у 2001 році [3]. Розробником програми MessagEase є Саїд Несбат (англ. Saied Nesbat) з каліфорнійської компанії EX-Ideas [4].

Методи одного натискання

Методи одного натискання (англ. «Single-Press Methods») – методи введення тексту, які направлені на зменшення кількості натискань, необхідних для введення одного символу, до одиниці [3]. Такі методи, на відміну від методів багатьох натискань, є багатозначними, оскільки послідовність натискань клавіш (англ. «key sequence») при введенні тексту в даному випадку може відповідати одночасно декільком словам у словнику. Фактично, завданням методів одного натискання є вирішення цієї багатозначності, яке здійснюється за допомогою використання словників.

Метод предиктивного введення тексту

Метод предиктивного введення тексту (англ. «The Predictive Text Entry Method») – метод введення тексту, що передбачає натискання однієї клавіші користувачем для введення кожного символу, здійснюючи при цьому співставлення введеної послідовності словам, що містяться у словнику [3]. Хоча і назва методу передбачає вгадування деяким чином

слова, яке збирається ввести користувач, фактично завданням методу є скоріше вибір деякого слова серед можливих варіантів слів, що відповідають певній послідовності натискань клавіш, аніж безпосередньо вгадування цього слова. Незважаючи на те, що одній і тій самій клавіші відповідає декілька літер, в багатьох випадках певній послідовності натискань клавіш відповідає лише одне слово зі словника. Таким чином, можливим є досягнення показників кількості натискань, необхідних для введення одного символу, близьких до одиниці. В разі, якщо деяка послідовність натискань клавіш відповідає двом або більше словам зі словника, користувач має можливість переглянути список слів, що відповідають даній послідовності, і обрати слово, яке він мав намір ввести.

Розглянемо приклад введення англійського слова “come” за допомогою методу предиктивного введення тексту. Спочатку користувач натискає клавішу із позначенням цифри “2”, оскільки перша літера слова “come” – “c” розташована саме на цій клавіші. Згідно стандарту ITU E 1.161, окрім літери ”c”, на клавіші із позначенням цифри “2” також розташовані літери “a” та “b”. Після натискання цієї клавіші користувачеві буде одразу запропоновано до введення слово “a”, оскільки таке слово відповідає послідовності натискань, що на даний момент складається з одного натискання клавіші із позначенням цифри “2”. Однак запропоноване слово не є тим словом, яке має намір ввести користувач, тому він продовжує введення символів. Користувач натискає клавішу із позначенням цифри “6”, оскільки друга літера слова “come” – “o” розташована саме на цій клавіші. Після цього послідовність натискань клавіш складатиме два елементи – клавіші із позначенням цифр “2” та “6”. Користувачеві буде запропоновано до введення слово “an”, оскільки воно відповідає даній послідовності, однак це слово також не є тим словом, яке має намір ввести користувач, тому він продовжує введення символів. Користувач знову натискає клавішу із позначенням цифри “6”, оскільки третя літера слова “come” – “m” так само, як і літера “o”, розташована на

цій клавіші. Користувачеві буде запропоновано до введення слово “an”, оскільки воно відповідає даній послідовності, однак це слово також не є тим словом, яке має намір ввести користувач, тому він продовжує введення символів. Нарешті, користувач натискає клавішу із позначенням цифри “3”, на якій розташована остання літера слова “come” – “e”. Користувачеві буде запропоновано до введення слово “come”, оскільки воно відповідає даній послідовності натискань. Слід зазначити, що крім слова “come”, заданій послідовності відповідають також слова “bone”, “bond” та інші. Такі слова також додаються до списку пропозицій для введення, і користувач має можливість вибрати необхідне слово з цього списку.

Також, слід зазначити, що ефективність даного методу може варіюватись в залежності від мови, на якій здійснюється введення тексту. Це пов’язано із тим, що середня довжина слів у різних мовах є різною. Так, наприклад, середня довжина слова в українській мові становить 7,86 літер, у англійській – 8,23 літер, у російській – 9,97 літер, у французькій – 10,09 літер, а у німецькій – 11,66 літер [5].

Однак, слід зазначити, що наведені дані про середню довжину слова у різних мовах світу не враховують частоти використання кожного з цих слів. Так, наприклад, англійське слово “the” використовується часто, тоді як таке слово, як “lugubrious” використовується рідко. Проте обидва слова однаково враховуються при підрахунку середньої довжини слова. Тому, при підрахунку середньої довжини слова також доцільним є використання текстових корпусів замість звичайного списку слів тієї чи іншої мови. Так, наприклад, середня довжина слова у англійській мові, підрахована за допомогою використання деяких текстових корпусів, складає від 4,23 до 5,50 літер [6 – 10]. Порівняння значень середньої довжини слова у англійській мові, отриманих з текстових корпусів та зі списку слів, наведено на рис. 2.

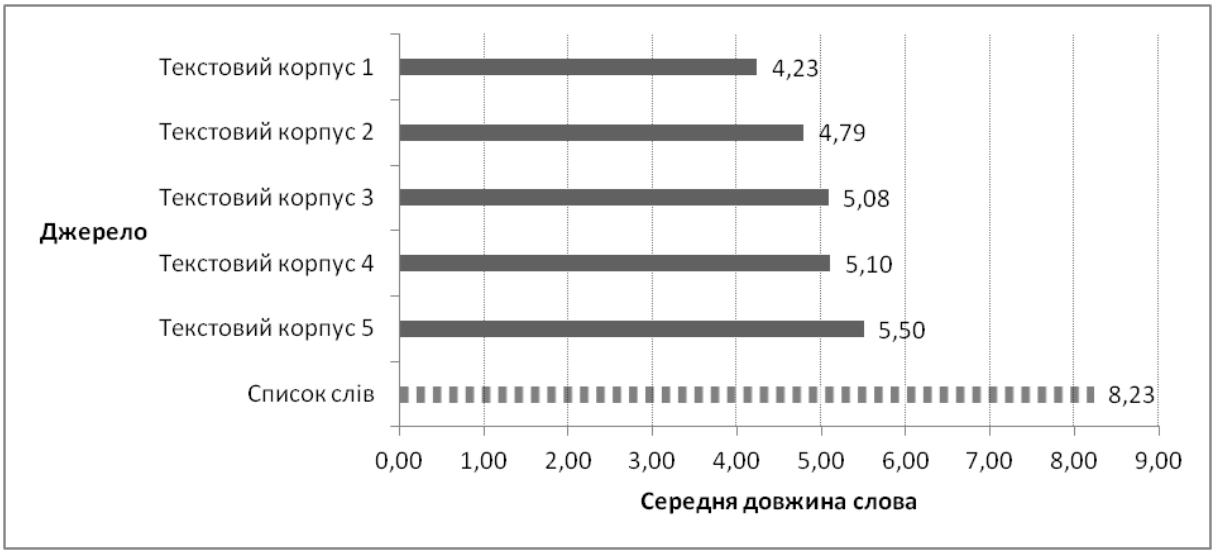


Рис. 2. Значення середньої довжини слова у англійській мові, отримані з текстових корпусів та зі списку слів

Залежність ефективності даного методу введення тексту від середньої довжини слова полягає у тому, що зі збільшенням довжини слова, що вводиться, зменшується кількість слів, що відповідають заданій послідовності натискань клавіш. Відповідно, чим меншою є кількість слів, що відповідають заданій послідовності натискань клавіш, тим меншою є кількість слів у списку пропозицій для введення, а відтак, тим вищою є ймовірність того, що користувачеві буде одразу запропоноване саме те слово, яке він мав намір ввести.

Метод предиктивного введення тексту використовується при введенні тексту на багатьох мобільних пристроях, укомплектованих 12-кнопковою клавіатурою. Найбільш поширеною реалізацією даного методу є програма T9, розроблена американським інженером, винахідником та підприємцем Кліффом Кушлером (англ. Cliff Kushler) з компанії Tegic Communications [11]. Першим мобільним телефоном із підтримкою T9 став мобільний телефон Sagem MC 850, який з'явився на ринку у 1999 році [12]. Також існують такі реалізації методу предиктивного введення тексту, як програма eZiText, створена компанією Zi Corporation, та програма iTap, створена компанією Motorola [3].

Більшість реалізацій методу предиктивного введення тексту здійснюють співставлення лише слів, довжина яких дорівнює довжині послідовності натискань клавіш, в результаті чого кількість натискань, необхідних для введення одного символу, є дещо вищою за одиницю при введенні слів, що містяться у словнику. При введенні ж слів, що відсутні у словнику кількість натискань, необхідних для введення одного символу, стає ще вищою.

Однак, деякі реалізації пропонують користувачеві для введення слова, довжина яких є більшою за довжину послідовності натискань клавіш. Це досягається за рахунок використання значень ймовірності введення того чи іншого слова. Такі реалізації дозволяють досягти кількості натискань, необхідних для введення одного символу, меншої за одиницю.

Метод WordWise

Метод WordWise – метод предиктивного введення тексту, розроблений компанією Eatoni Ergonomics [3]. Даний метод передбачає використання допоміжної клавіші. Згідно даного методу, вибір літери, розташованої на деякій клавіші, здійснюється шляхом одночасного натискання цієї клавіші, а також допоміжної клавіші, за допомогою якої вказується позиція літери на клавіші. Застосування методу WordWise сприяє суттєвому зменшенню кількості слів, що відповідають послідовності натискань, оскільки користувач явно вказує деякі символи послідовності, що вводиться. Однак, даний метод має суттєвий недолік, який полягає у тому, що при введенні тексту необхідним є натискання двох клавіш одночасно. Враховуючи обмежені розміри клавіатури мобільних пристроїв, одночасне натискання двох клавіш може бути достатньо складним для деяких користувачів.

Метод LetterWise

Метод LetterWise – метод предиктивного введення тексту, також запропонований компанією Eatoni Ergomomics [3]. Даний метод дозволяє

здійснювати введення тексту без використання великих словників. Так, під час введення тексту за допомогою даного методу розглядаються лише значення ймовірностей появи однієї літери після іншої. Так, в англійській мові після літери “t” у слові часто зустрічається літера “h”, однак рідко зустрічається літера “g”. Отже, під час застосування даного методу відбувається вибір найбільш ймовірної літери, яка може розташовуватися у слові після попередньої. Користувач має можливість переглядати символи, що йому пропонуються, та обирати той, який йому потрібен.

За допомогою використання даного методу можливим є досягнення достатньо невеликої кількості натискань, необхідних для введення одного символу. Крім того, одним з основних переваг даного методу є достатньо невеликий обсяг пам’яті, необхідний для роботи даного методу, оскільки у ній зберігаються лише дані про ймовірності деякої літери після іншої. Також, даний метод дозволяє вводити слова, що не містяться у словнику, так само ефективно, як і ті, що містяться у ньому. Враховуючи це, метод LetterWise може використовуватися як допоміжний метод у поєднанні із іншими, більш складними методами, замість методів багатьох натискань, оскільки сприяє вищій швидкості введення.

Методи предиктивного введення тексту на основі N-грам

Методи предиктивного введення тексту, розглянуті вище, під час своєї роботи здійснюють прогнозування лише слова, що вводиться користувачем в поточний момент. Для цього застосовуються словники, дані про частоту використання тих чи інших літер, дані про частоту появ одних літер після інших тощо. Однак, при цьому не застосовуються дані про частоту появ одних слів після інших.

Методи предиктивного введення тексту на основі *N*-грам (англ. «Predictive Text Entry Using *N*-grams») дозволяють здійснювати прогнозування наступного слова на основі попередніх слів [3]. Це досягається за рахунок використання даних про ймовірності появи одних

слів після інших, отриманих, як правило, в результаті оброблення текстових або мовленнєвих корпусів відповідної мови.

Текстовий корпус (англ. «text corpus») – спеціальним чином підібрана та оброблена за певними правилами сукупність текстів, складених певною мовою, що використовується для наукових досліджень [13]. В свою чергу, мовленнєвий корпус (англ. «speech corpus») є різновидом текстового корпусу, що містить дані аудіофайлів та транскрипцій текстів [14].

N-грамою (англ. «*N*-gram») називають деяку послідовність, що складається з N елементів [15]. Елементами такої послідовності можуть бути, наприклад, звуки, літери, склади або слова. На практиці зазвичай використовуються *N*-грами літер або слів. Послідовність слів, що формує стійке словосполучення, називають колокацією (англ. «collocation») [16].

Іншими словами, до складу *N*-грами входять елементи, що розташовуються одне за одним у деякому наборі таких елементів, наприклад, послідовність літер у слові або послідовність слів у тексті.

N-грама, що складається з одного елементу, називається також уніграмою (англ. «unigram») [17]. *N*-грама, що складається з двох елементів, називається також біграмою (англ. «bigram», «digram»). В свою чергу, *N*-грама, що складається з трьох елементів, називається також триграмою (англ. «trigram»). Крім того, іноді застосовуються також числові позначення для *N*-грам, що складаються з більше, ніж трьох елементів, наприклад 4-грама (англ. «four-gram»), 5-грама (англ. «five-gram») і так далі.

Позначення *N*-грам відповідно до кількості елементів, з яких вони складаються, наведено у табл. 1.

Приклад процесу створення уніграм, біграм та триграм слів за допомогою використання деякого текстового корпусу англійської мови наведено на рис. 3.

Таблиця 1

Позначення *N*-грам відповідно до кількості елементів, з яких вони складаються

Кількість елементів *N*-грами	Позначення
1	уніграма
2	біграма
3	триграма
4	4-грама
5	5-грама
…	…

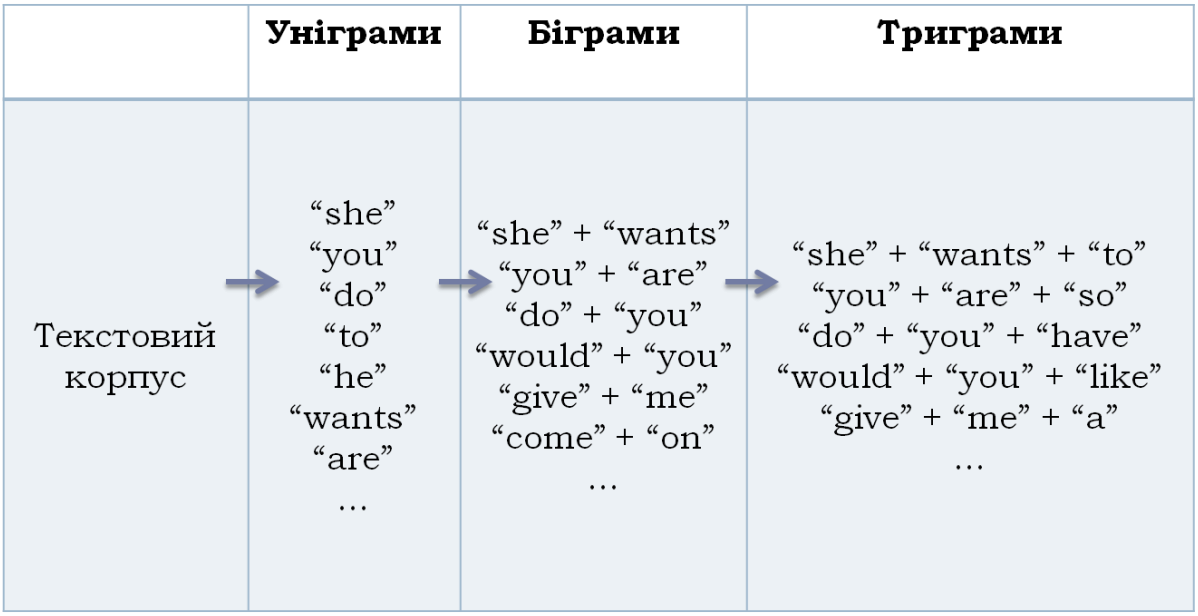


Рис. 3. Приклад процесу створення уніграм, біграм та триграм слів за допомогою використання деякого текстового корпусу англійської мови

Метою створення уніграм є визначення ймовірностей появи того чи іншого елемента в певному наборі таких елементів, наприклад, визначення частоти використання тієї чи іншої літери деякого алфавіту. На рис. 4 наведено частотний розподіл використання літер у англійській мові. Кожна літера у тексті в даному випадку є уніграмою, а відсоток від загальної

кількості уніграм, відповідно, показує ймовірність появи тієї чи іншої уніграми у тексті.

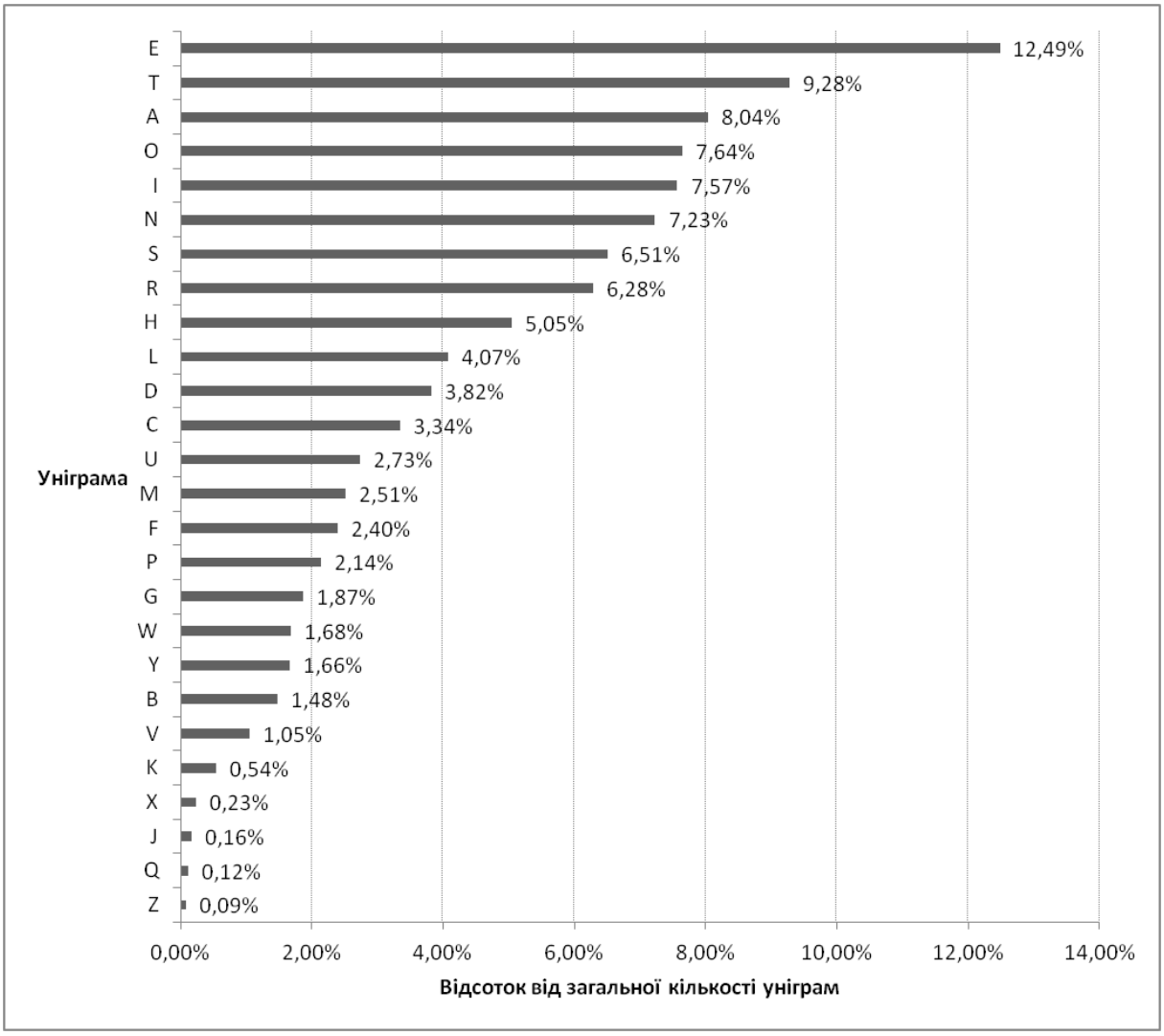


Рис. 4. Частотний розподіл уніграм літер англійської мови

Метою створення інших *N*-грам є визначення ймовірностей появи тієї чи іншої комбінації елементів, або, іншими словами, ймовірностей появи одного елемента після іншого, в певному наборі таких елементів. Прикладом може слугувати визначення ймовірностей появи однієї літери після іншої у слові, або ж одного слова після іншого у тексті. Так, значення ймовірностей появи послідовностей двох літер, або, іншими словами, їх біграм, використовуються під час роботи деяких методів предиктивного введення тексту, таких як, наприклад, метод LetterWise, розглянутий вище.

На відміну від кількості уніграм літер певної мови, що є рівною кількості літер у цій мові, кількість біграм для цієї мови становить вже $N*N$ елементів, де N – кількість уніграм, або літер цієї мови. Відповідно, кількість триграм становитиме вже $N*N*N$ елементів. Таким чином, формула для підрахунку кількості N-грам виглядатиме наступним чином:

$$Q(N) = Q(1)^N = U^N \text{ при } N \geq 1,$$

де $Q(N)$ – кількість N-грам,

N – кількість елементів послідовності,

U – кількість уніграм.

Отже, кількість уніграм для англійської мови становитиме 26 елементів, а кількість біграм, відповідно, 676 елементів. Частотний розподіл десяти найбільш часто вживаних біграм літер для англійської мови наведено на рис. 5 [7].

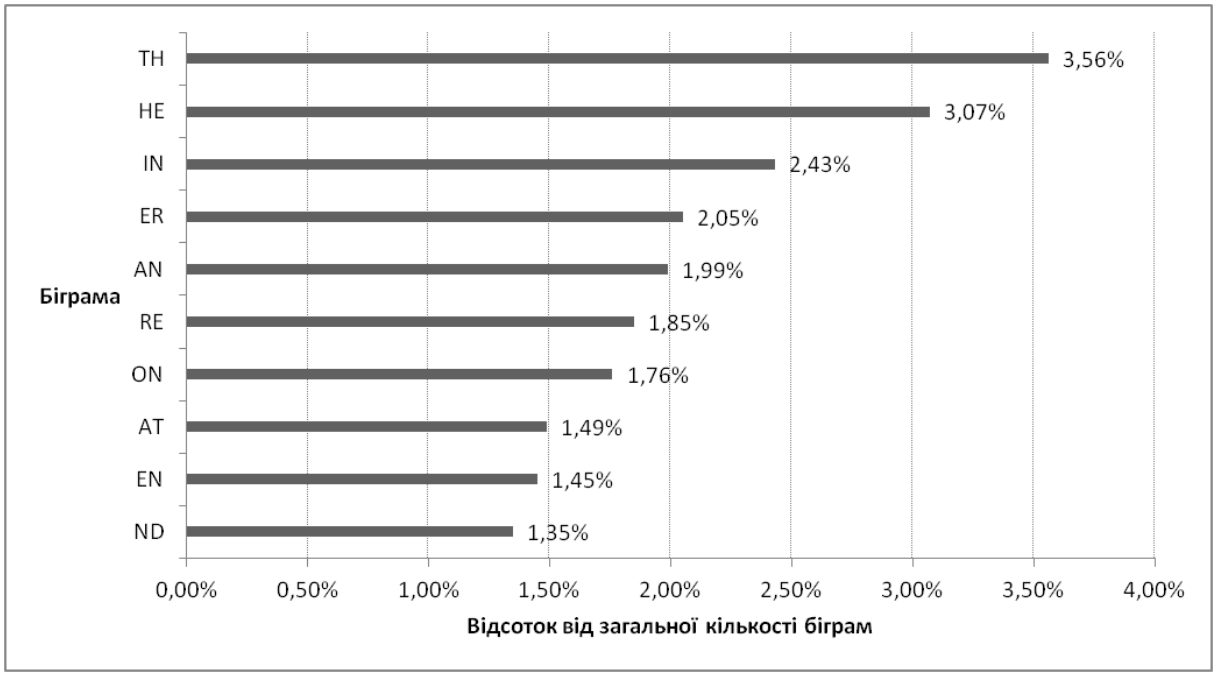


Рис. 5. Частотний розподіл десяти найбільш часто вживаних біграм літер англійської мови

Частотний розподіл п’яти найбільш часто вживаних N-грам літер англійської мови для значень N від 1 до 5 наведено у табл. 2 [7].

Таблиця 2

Частотний розподіл п’яти найбільш часто вживаних *N*-грам літер англійської мови для значень *N* від 1 до 5

N=1	*N*=2	*N*=3	*N*=4	*N*=5
e	th	the	tion	ation
t	he	and	atio	tions
a	in	ing	that	which
o	er	ion	ther	ction
i	an	tio	with	other

Отже, таким чином, видно, що після деякої введеної літери ймовірність появи одних літер може бути високою, а ймовірність появи інших – низькою. В свою чергу, різною може бути і ймовірність появи різних літер після деякої комбінації з двох літер, після комбінації з трьох літер і так далі. Відповідно, в залежності від значень цих ймовірностей, методи предиктивного введення тексту можуть пропонувати користувачеві ті чи інші літери або слова.

На відміну від методів предиктивного введення тексту, що використовують для своєї роботи значення ймовірностей появи *N*-грам літер, даний метод передбачає використання значень ймовірностей появи *N*-грам слів. Процес створення таких *N*-грам на основі деякого текстового корпусу здійснюється аналогічно до процесу створення *N*-грам літер.

За допомогою даного методу можливим є прогнозування наступного слова одразу після закінчення введення попереднього. Це, в свою чергу, дозволяє у деяких випадках забезпечувати введення слова за допомогою лише одного натискання. В результаті, середнє значення кількості натискань, необхідних для введення одного слова, може навіть досягати значень, менших за одиницю.

1.3. Аналіз існуючих програмних засобів, що реалізують методи введення тексту на мобільних пристроях

Враховуючи те, що більшість існуючих методів введення тексту були розроблені ще за часів широкого використання мобільних пристроїв із 12-кнопковими клавіатурами, велика кількість програмних засобів, що реалізують ці методи, також розроблялась у ті часи.

Однак, деякі програмні засоби можуть застосовуватись також і на сучасних мобільних пристроях із сенсорними екранами, введення тексту на яких здійснюється за допомогою віртуальної екранної клавіатури.

1.3.1. MessagEase

Програма MessagEase є однією з реалізацій методу перевизначеної клавіатури. В основі цієї програми лежить клавіатура у вигляді матриці розміром 3x3. Користувачі можуть здійснювати на ній такі дії, як натискання та проведення (англ. «swipe») вверх, вниз, ліворуч, праворуч або по діагоналі для доступу до всіх клавіш та символів [4]. Зображення клавіатури програми MessagEase наведено на рис. 6.

Рис. 6. Зображення клавіатури програми MessagEase

Введення найбільш часто вживаних літер здійснюється за допомогою натискання. Введення літер, що вживаються дещо рідше, здійснюється за

допомогою проведення. Так, наприклад введення літери “o” відбувається за допомогою натискання центрального квадрату. В свою чергу, введення літери “c” відбувається за допомогою проведення від центрального квадрату ліворуч. При проведенні на екрані також відображається зелена лінія, що показує шлях проведення пальцем. Програма підтримує користувацькі словники, які використовуються для прогнозування та виправлення слів.

Отже, дев’ять найбільш часто вживаних літер англійської мови – “e”, “t”, “a”, “o”, “n”, “r”, “i”, “s” та “h”, розміщуються на клавіатурі таким чином, що їх введення можливе за допомогою одного натискання. Наступні за частотою вживання 17 літер: “d”, “l”, f”, “c”, “m”, “u”, “g”, “y”, “p”, “w”, “b”, “v”, “k”, “j”, “x”, “q” та “z” розміщуються таким чином, що їх введення можливе за допомогою проведення від або до центральної клавіші із позначенням літери “o”, за винятком літери “z”, що розташована біля клавіші із позначенням літери “e” разом із деякими розділовими знаками. Так, наприклад, введення літери “v” здійснюється за допомогою проведення пальцем від літери “a” до літери “o”, а введення літери “d” – за допомогою проведення пальцем від літери “o” до літери “e”.

Розглянемо приклад введення англійського слова “dog” за допомогою програми MessagEase. Так, введення слова “dog” здійснюється за допомогою послідовності, що складається з проведення пальцем від клавіші із позначенням літери “o” до клавіші із позначенням літери “e”, натискання клавіші із позначенням літери “o”, а також проведення пальцем від клавіші із позначенням літери “o” до клавіші із позначенням літери “t”.

Також, на екрані міститься панель, за допомогою якої користувачеві надається доступ до стандартних операцій вирізання (англ. «cut»), копіювання (англ. «copy») та вставлення (англ. «paste»), цифрової клавіатури, зміни регістру та інших клавіш управління. Доступ до таких функцій здійснюється також за допомогою проведення пальцем від однієї клавіші до іншої.

На даний момент програма доступна для пристроїв, що працюють під управлінням операційних систем Android та iOS, а також для Apple Watch. Програма на даний момент підтримує 19 мов, в тому числі українську та російську.

1.3.2. T9

Програма T9 є найбільш поширеною реалізацією методу предиктивного введення тексту. T9 використовується здебільшого на мобільних пристроях, укомплектованих 12-кнопковою клавіатурою розміром 3x4 клавіші [18]. Назва “T9” розшифровується як “Текст на 9 клавішах” (англ. «Text on 9 keys»).

T9 використовується на мобільних телефонах брендів Nokia, Samsung, Siemens, Sony Ericsson, Sagem та інших, а також на деяких кишенькових персональних комп’ютерах.

Після появи та широкого розповсюдження смартфонів T9 перестала широко використовуватись, оскільки на нових смартфонах почали встановлюватись сенсорні дисплеї, що, відповідно, містили екранні клавіатури. Однак, T9 використовується і сьогодні на деяких недорогих мобільних телефонах, що не мають сенсорного дисплею.

T9 дозволяє введення слів за допомогою здійснення одного натискання клавіші для введення однієї літери, що є суттєвим вдосконаленням у порівнянні із методами багатьох натискань, що були на час розроблення T9 основними методами введення тексту та відповідно до яких, введення однієї літери часто потребувало кількох натискань.

T9 поєднує групи літер, що розміщені на кожній клавіші на клавіатурі мобільного телефону, із словником. Після цього відбувається пошук у словнику всіх слів, що відповідають даній послідовності натискань клавіш, та їх сортування за частотою використання. Також, в процесі ознайомлення програми зі словами та фразами, які користувач часто використовує, процес введення тексту прискорюється за рахунок того, що користувачеві спочатку пропонуються для введення слова, які він

найчастіше використовує, а вибір інших варіантів здійснюється за допомогою одного чи декількох натискань клавіші “Next” (англ. «next» – наступний).

Найчастіше T9 використовується на мобільних телефонах, що укомплектовані 12-кнопковою клавіатурою, розташування літер англійської мови на якій відповідає стандарту ITU E 1.161.

1.3.3. iTap

Програма iTap була розроблена компанією Motorola в якості конкурента програмі T9 [19]. При послідовному введенні трьох або більше символів, iTap намагається вгадати закінчення слова. Наприклад, при введенні послідовності “prog” користувачеві буде запропоновано слово “program”. В разі, якщо користувач мав намір ввести інше слово, наприклад, “progress”, або слово, що складається з інших початкових літер, але таких, що відповідають тій самій послідовності натискань клавіш, наприклад, “prohibited” або “spoil”, користувач має можливість натиснути клавішу із позначенням стрілки для виділення інших слів з меню для вибору, що відсортовані за частотою використання.

Після введення слова автоматично ставиться пробіл. У деяких реалізаціях натискання клавіші введення пробілу, що зазвичай є клавішою із позначенням символу зірочки (“*”), призводить до введення поточної послідовності символів, не підставляючи при цьому закінчення запропонованого слова.

Так само, як і остання версія програми T9, програма iTap також має можливість закінчення слів та фраз. Це досягається за рахунок того, що у словнику, окрім слів, містяться також і фрази та часто вживані речення. Таким чином, пропозиції для введення можуть залежати від контексту, у якому перебуває слово, що вводиться в поточний момент.

1.3.4. Пошуковий сервіс Google

Одним з найяскравіших прикладів програмних засобів, що при введенні тексту користувачем здійснюють прогнозування наступних слів, є

пошуковий сервіс Google [20]. Так, в процесі введення користувачем тексту у полі для запиту під час підготовки до здійснення пошукового запиту за допомогою пошукового сервісу Google, користувачеві автоматично надаються пропозиції для пошуку, що базуються на попередніх запитах цього конкретного користувача, а також на найбільш популярних запитах користувачів загалом, що починаються із вже введеної комбінації символів. При створенні списку пропозицій для пошуку може також враховуватися місцезнаходження користувача. Приклад відображення поля для запиту при введенні користувачем слова "київський" із пропозиціями для введення наведено на рис. 7.

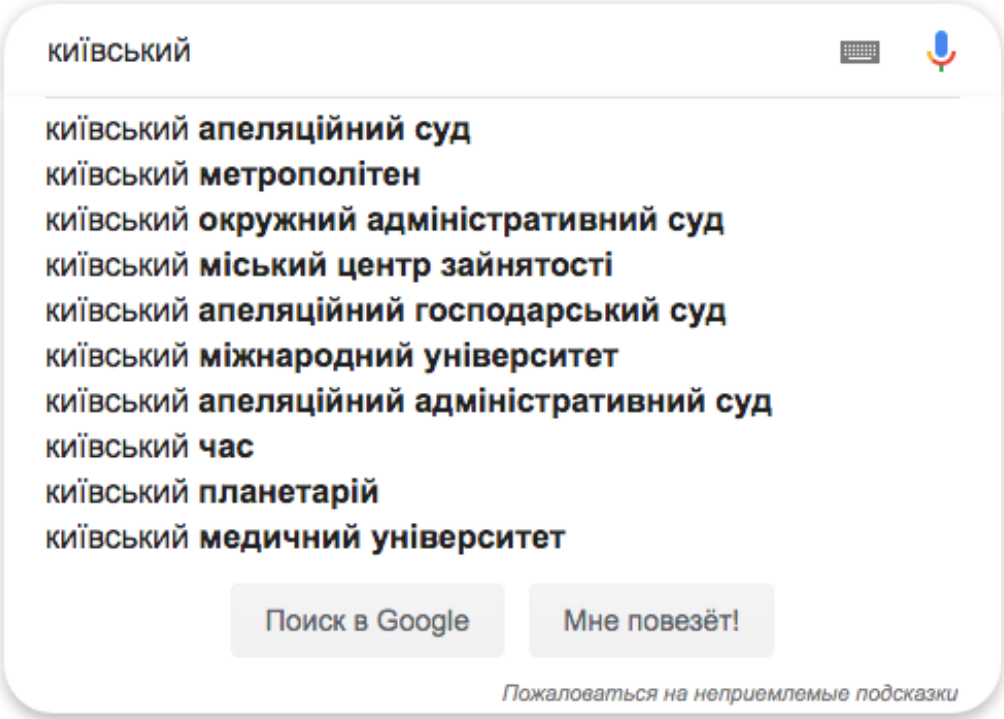


Рис. 7. Приклад відображення поля для запиту пошукового сервісу Google із пропозиціями для введення при введенні користувачем слова "київський"

Слід зазначити, що принцип роботи програмних засобів, що надають можливість предиктивного введення пошукових запитів за допомогою пошукового сервісу Google, є однаковим як для настільних комп'ютерів та

ноутбуків, так і для мобільних пристроїв. Однак, слід зазначити, що предиктивне введення тексту за допомогою даного сервісу призначене безпосередньо для подальшого здійснення пошукового запиту, і не призначене для введення тексту при спілкуванні користувачів між собою.

1.4. Особливості ефективності існуючих засобів предиктивного введення тексту на мобільних пристроях

Існуючі засоби предиктивного введення тексту на мобільних пристроях демонструють достатньо високу якість роботи лише за певних умов. Такими умовами можуть бути, наприклад, відсутність у тексті, що вводиться користувачем мобільного пристрою, орфографічних помилок або випадкових натискань сусідніх клавіш. Також, в процесі введення тексту, як це трапляється достатньо часто, користувач може поспішати, і випадково натиснути необхідну клавішу зайвий раз або не натиснути її взагалі.

Крім того, деякі засоби предиктивного введення тексту не здатні пристосовуватися до особливостей введення тексту тим чи іншим користувачем. Наприклад, що є особливо поширеним у наш час, під час відправлення текстових повідомлень за допомогою різноманітних сервісів, що надають послуги з обміну текстовими повідомленнями, та соціальних мереж, користувач може застосовувати під час написання повідомлень специфічну розмовну лексику. Велика кількість слів, що присутня у таких повідомленнях, може не міститися у звичайних словниках, а відтак, засоби предиктивного введення, встановлені на цьому пристрої, можуть бути не здатні оброблювати такі слова так само ефективно, як інші слова, що містяться у словнику, оскільки вважатимуть, що користувач ввів помилкове слово. В першу чергу, це стосується різноманітних скорочень, які з урахуванням специфіки введення тексту на мобільних пристроях, де важливими є висока швидкість введення тексту та малий розмір кінцевого

повідомлення, використовуються дуже часто. Переважна більшість таких скорочень не є літературними і не міститься у словниках.

Особливо популярними є так звані інтернет-акроніми – абревіатури, що із розповсюдженням інтернету та поширенням спілкування за допомогою нього стали усталеними у інтернет-середовищі, причому деякі з них згодом навіть вийшли за його межі. Однак велика кількість таких акронімів залишається відсутньою у словниках, тим паче, що навіть ті, які опиняються у словнику, все одно стають доступними для засобів предиктивного введення тексту на мобільних пристроях набагато пізніше, ніж отримують широке розповсюдження. Це пов'язано із тим, що, по-перше, слова додають у словники лише через певний час після початку їх широкого застосування, і, по-друге, словники, які використовують для своєї роботи засоби предиктивного введення тексту на мобільних пристроях, оновлюються теж лише через певний проміжок часу.

Також, користувачами сучасних засобів обміну повідомленнями широко застосовуються слова, що навмисно неправильно перекладені з інших мов, містять навмисні помилки або такі, що представлені у формі, що відрізняється від їх літературного написання. Ймовірність потрапляння таких слів у словник є ще нижчою, аніж у широко вживаних скорочень.

На перший погляд, може здаватися очевидним той факт, що засоби предиктивного введення тексту не повинні жодним чином враховувати під час своєї роботи слова, написані з помилками, або скорочення, що не є літературними і не містяться у словниках. Іншими словами, вони повинні опрацьовуватися так само, як і засобами перевірки орфографії, тобто розглядатися такими, що написані неправильно і не повинні бути представлені у якості можливих пропозицій для введення.

Однак, на відміну від засобів перевірки орфографії, для яких основною задачею є повідомлення користувачеві про наявність або відсутність у слові помилок, тобто його правильність або неправильність написання, засоби предиктивного введення тексту в першу чергу повинні

забезпечувати високу швидкість введення. Для досягнення цієї мети необхідним є пристосування таких засобів до особливостей введення тексту користувачем, до яких, в тому числі, може належати повсякденне вживання слів і скорочень, що відсутні у словнику. Тому в даному випадку необхідним є запам’ятовування і додавання таких слів у словник та навіть надання таким словам під час генерування списку пропозицій для введення вищого пріоритету, аніж тим словам із початкового встановленого словника, що схожі за написанням, але рідше застосовуються цим користувачем.

Таким чином, запам’ятовування навіть тих слів, що відсутні у словнику, однак які використовує користувач у повсякденному спілкуванні за допомогою текстових повідомлень, сприяє пристосуванню до особливостей його лексики, що, в свою чергу, суттєво підвищує ефективність застосування засобів предиктивного введення тексту, оскільки позитивно впливає на вирішення їх основної задачі – забезпечення швидкого введення тексту на мобільному пристрої.

Отже, проаналізувавши існуючі методи введення тексту на мобільних пристроях, а також програмні засоби, що реалізують ці методи, можна зробити висновок, що методи предиктивного введення тексту повинні мати можливість пристосування до особливостей введення тексту тим чи іншим користувачем, використовуючи вже введені ним слова під час набирання наступних. Крім того, враховуючи те, що під час швидкого введення, яке є достатньо поширеним явищем, в умовах обмежених розмірів екранних клавіатур на сучасних мобільних пристроях достатньо ймовірним є помилкове натискання сусідньої клавіші, методи предиктивного введення тексту на мобільних пристроях повинні при створенні списку пропозицій для введення також враховувати можливість таких помилкових натискань.

2. ОПИС РІШЕННЯ, ЩО ПРОПОНУЄТЬСЯ

2.1. Критерії ефективності методів введення тексту на мобільних пристроях

Для визначення ефективності того чи іншого методу введення тексту та порівняння ефективності різних методів між собою необхідно обрати критерій ефективності.

2.1.1. Критерій KSPC

Одним з найбільш поширених критеріїв ефективності методів введення тексту є критерій KSPC (англ. «Keystrokes per Character»), що показує середню кількість натискань клавіш на клавіатурі, необхідних для введення одного символу [3].

Значення критерію KSPC для деякого слова w можна обчислити так:

$$KSPC_w = \frac{K_w}{C_w},$$

де K_w – кількість натискань, необхідних для введення слова w,

C_w – кількість символів у слові w.

Відповідно, загальне значення критерію KSPC при введенні тексту можна обчислити таким чином:

$$KSPC = \frac{\sum(K_w)}{\sum(C_w)}.$$

Значення критерію KSPC для методів багатьох натискань є більшим за одиницю, оскільки введення кожного символу зазвичай потребує декількох натискань. Для методів одного натискання значення KSPC є в цілому меншим, оскільки введення символу може в деяких випадках здійснюватися за допомогою лише одного натискання. Введення тексту за допомогою клавіатури настільних комп’ютерів та ноутбуків, в свою чергу, забезпечує значення KSPC рівним одиниці, оскільки одному символу відповідає одне натискання. Слід зазначити, що в даному випадку до уваги беруться натискання клавіш без урахування спеціальних символів та літер у верхньому регістрі тощо, які потребують натискання додаткових клавіш.

Методи ж предиктивного введення тексту на основі N-грам дозволяють досягати значень KSPC, менших за одиницю, оскільки у деяких випадках введення слова може здійснюватися за допомогою натискань, кількість яких є меншою за кількість літер у цьому слові.

2.1.2. Критерій CPS/CPM

Іншим підходом до визначення ефективності того чи іншого методу введення тексту є вимірювання кількості слів, введених за певний час. Для цього можна застосовувати такий показник, як CPS (англ. «Characters per Second»), що показує кількість символів, введених за секунду [21].

Значення критерію CPS для деякого слова w можна обчислити наступним чином:

$$CPS_w = \frac{C_w}{t_w},$$

де C_w – кількість символів у слові w,

t_w – час, необхідний для введення слова w (у секундах).

Відповідно, загальне значення критерію CPS при введенні тексту можна обчислити наступним чином:

$$CPS = \frac{\sum(C_w)}{\sum(t_w)}.$$

Також, при підрахунку швидкості введення тексту може застосовуватись критерій CPM (англ. «Characters per Minute»), що показує кількість символів, введених за хвилину.

Значення критерію CPM зі значення критерію CPS можна отримати наступним чином:

$$CPM = CPS * 60.$$

Відповідно, значення критерію CPS можна отримати зі значення критерію CPM наступним чином:

$$CPS = \frac{CPM}{60}.$$

Крім того, значення критерію CPM можна отримати аналогічно до критерію CPS із відмінністю у тому, що значення t_w повинно вимірюватися у хвилинах.

2.1.3. Вибір критерію для визначення ефективності методів предиктивного введення тексту

Слід зазначити, що критерій CPS/CPM, розглянутий вище, може варіюватися в залежності від навичок введення тексту того чи іншого користувача. Це пов'язано із тим, що різні користувачі можуть вводити текст із різною швидкістю. Так, розподіл значень CPM для різних користувачів наведено на рис. 8 [22].

Рис. 8. Розподіл значень CPM для різних користувачів

Як видно, діапазон значень CPM може бути достатньо широким. Відповідно, порівняння між собою методів введення тексту за допомогою критерію CPS/CPM, отриманого при введенні тексту різними людьми може призвести до некоректного результату, оскільки не враховуватиметься швидкість введення тексту тим чи іншим користувачем.

В свою чергу, критерій KSPC, також розглянутий вище, дозволяє визначати ефективність методів введення тексту незалежно від швидкості введення тексту тим чи іншим користувачем, оскільки враховує не час, необхідний на натискання клавіш, а безпосередньо кількість цих натискань.

Отже, в якості критерію ефективності методів введення тексту використовуватимемо критерій KSPC.

2.2. Обґрунтування вибору методу для модифікації

Методи багатьох натискань не забезпечують можливості прогнозування слів, і, відповідно, потребують більше одного натискання для введення одного символу. Таким чином, методи багатьох натискань не дозволяють досягати значень KSPC, менших за одиницю, тому обрання таких методів для модифікації є недоцільним.

Методи одного натискання, що використовують для своєї роботи частотні розподіли літер алфавіту, і, відповідно, можуть здійснювати прогнозування наступної літери у слові, дозволяють суттєво підвищити ефективність введення тексту у порівнянні із методами багатьох натискань. Відповідно, значення критерію KSPC для методів одного натискання в цілому є нижчим, ніж для методів багатьох натискань.

Проте, враховуючи специфіку введення тексту на мобільних пристроях, достатньо часто може траплятися ситуація, коли користувач має намір ввести слово, що відсутнє у словнику. В такому випадку це слово не буде міститися у списку слів, що пропонуються користувачеві для введення. Для того, щоб надати можливість користувачеві ввести слово, що відсутнє у словнику, необхідним є застосування звичайного методу багатьох натискань у якості допоміжного методу до методу одного натискання. Такий допоміжний метод забезпечить можливість однозначного введення необхідного слова користувачем у такій ситуації. Відповідно, слід також забезпечити можливість переключення між цими

методами введення за допомогою, наприклад, спеціальної клавіші на клавіатурі або додавання окремого пункту у меню.

Однак, така висока залежність від словника та, відповідно, необхідність переключення між методами введення при відсутності слова, що має намір ввести користувач, у словнику, може суттєво вплинути на швидкість введення тексту за допомогою даного методу. Так, фактично, при введенні слова, що не міститься у словнику, користувачеві, окрім часу, витраченого на натискання відповідних клавіш, необхідно також витратити час на переключення методу введення тексту та безпосередньо введення необхідного слова за допомогою методу багатьох натискань. Тому швидкість введення тексту за допомогою даного методу при частому введенні слів, що відсутні у словнику, може бути достатньо низькою. Можливим шляхом вирішення даної проблеми може бути, наприклад, запам’ятовування слів, що були введенні користувачем вручну, і, відповідно, додавання їх до словника, що, таким чином, забезпечить додавання цього слова у список пропозицій для введення при наступному введенні цього слова користувачем. Іншим шляхом вирішення даної проблеми може бути використання під час роботи методу значень ймовірностей розташування однієї літери після іншої у словах певної мови. Такий підхід застосовується, зокрема, у методі предиктивного введення тексту LetterWise.

Але, слід зазначити, що більшість методів одного натискання не дозволяє здійснювати введення слова за допомогою кількості натискань, що є меншою за кількість літер у слові. Відповідно, такі методи також не дозволяють досягати значень критерію KSPC, менших за одиницю. Ті ж методи одного натискання, що мають можливість прогнозування закінчень слова, і, відповідно, можуть у деяких випадках досягати значень критерію KSPC, менших за одиницю, не дозволяють здійснювати прогнозування наступного слова із врахуванням попереднього, за винятком деяких усталених фраз.

Для того, щоб знизити значення критерію KSPC ще більше, необхідним є використання таких методів, як методи предиктивного введення тексту на основі *N*-грам, які дозволяють здійснювати як прогнозування закінчення слова ще до його повного введення, після написання лише кількох перших літер цього слова, так і прогнозування наступного слова із врахуванням попереднього. Відповідно, у випадках, коли слово, які користувач має намір ввести після введення попереднього слова, міститься у списку для пропозицій для введення, такі методи дозволяють здійснювати введення слова за допомогою кількості натискань, необхідних для вибору слова зі списку пропозицій для введення. Враховуючи те, що сучасні мобільні пристрої зазвичай комплектуються сенсорними дисплеями, у списку пропозицій для введення може знаходитися декілька слів, які одночасно відображаються на екрані. В свою чергу, вибір такого слова може здійснюватися за допомогою лише одного натискання. Відповідно, оскільки в такому випадку введення слова може відбуватися за допомогою лише одного натискання, це дозволяє досягати для таких слів значень критерію KSPC, рівних $1/C_w$, де C_w – кількість літер у слові w. Причому, це значення буде зменшуватися із збільшенням довжини слова. Так, наприклад, для слова, що складається з 5 літер, значення KSPC становитиме 0,20, а для слова з 7 літер – лише 0,14.

Отже, в якості методу для модифікації використовуватимемо метод предиктивного введення тексту на основі *N*-грам.

2.3. Модифікація методу предиктивного введення тексту на мобільних пристроях на основі *N*-грам

Як було зазначено вище, методи предиктивного введення тексту на основі *N*-грам дозволяють здійснювати прогнозування наступного слова, що користувач має намір ввести, базуючись на даних про частоту використання тих чи інших комбінацій слів у текстах мови, на якій здійснюється введення. Такі комбінації або послідовності слів називають

N-грамами, де число N відповідає кількості слів у цій послідовності. При значенні N рівним одиниці, тобто при послідовності, що складається з одного елемента, яким в даному випадку є слово, така послідовність називається уніграмою. Тоді як значення ймовірностей використання послідовностей, що складаються з двох слів та більше, тобто N-грам при $N \geq 2$, використовуються для прогнозування наступного слова після введення попереднього, значення ймовірностей використання тієї чи іншої уніграми можна використовувати для прогнозування закінчення певного слова, початок якого вже введено користувачем.

2.3.1. Визначення ймовірності появи послідовності

Для визначення ймовірностей появи тієї чи іншої послідовності, або N-грами, у деякому текстовому корпусі, використовуватимемо метод максимальної правдоподібності [3].

Метод максимальної правдоподібності (англ. «MLE» – «maximum likelihood estimation») у математичній статистиці — це метод оцінювання невідомого параметра шляхом максимізації функції правдоподібності [23]. Він ґрунтується на припущенні про те, що вся інформація про статистичну вибірку міститься у функції правдоподібності. Метод максимальної правдоподібності був проаналізований, рекомендований і значно популяризований англійським вченим Рональдом Фішером (англ. Sir Ronald Aylmer Fisher) в період між 1912 і 1922 роками, хоча раніше також використовувався такими вченими, як Гаусс, Лаплас та інші. Оцінка максимальної правдоподібності є популярним статистичним методом, який використовується для створення статистичної моделі на основі даних і забезпечення оцінки параметрів моделі.

Метод максимальної правдоподібності відповідає багатьом відомим методам оцінки в області статистики. Наприклад, при розгляді деякого антропометричного параметру, такого як зростання кількості населення у деякій країні, за наявності даних про зростання деякої кількості людей, а не всього населення загалом, і припустивши, що зростання є нормально

розподіленою величиною з невідомою дисперсією і середнім значенням, середнє значення і дисперсія зростання у вибірці є максимально правдоподібними до середнього значення і дисперсії всього населення.

Для фіксованого набору даних і базової імовірнісної моделі, використовуючи метод максимальної правдоподібності, можна отримати значення параметрів моделі, які роблять дані «ближчими» до реальних. Оцінка максимальної правдоподібності дає можливість отримання рішення у достатньо простий та інтуїтивно зрозумілий спосіб.

Отже, використовуючи для створення N-грам текстові корпуси – набори текстів, статистично репрезентативні для моделювання деякої мови, можна отримати достатньо коректні значення ймовірностей появи N-грам при застосуванні методу максимальної правдоподібності.

Таким чином, згідно методу максимальної правдоподібності, ймовірність появи деякого слова w_n може бути визначена шляхом отримання кількості поява унірам $C(w_n)$ та їхньої нормалізації зі кількістю всіх унірам:

$$P(w_n) = \frac{C(w_n)}{\sum C(w)}.$$

В свою чергу, ймовірність появи слова w_n після слова w_{n-1} може бути визначена шляхом отримання кількості поява біграм $C(w_{n-1}, w_n)$ та їхньої нормалізації за кількістю всіх біграм, першим словом яких є слово w_{n-1}:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n)}{\sum C(w_{n-1}, w)}.$$

Оскільки кількість всіх біграм, що починаються зі слова w_{n-1}, дорівнює кількості унірам для цього слова, цей вираз можна спростити наступним чином:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n)}{C(w_{n-1})}.$$

Відповідно, в загальному випадку формула обчислення ймовірності появи довільної *N*-грами матиме вигляд:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}, w_n)}{C(w_{n-N+1}^{n-1})}.$$

2.3.2. Застосування розподілу літер між блоками

Слід зазначити, що при використанні методу предиктивного введення тексту на основі *N*-грам коректне співставлення слів зі словником відбувається тільки в тому разі, якщо користувач безпомилково натискає клавіші, що відповідають літерам відповідного слова.

Іншими словами, для того, щоб користувачеві було запропоноване слово, яке він має намір ввести, необхідним є точний збіг всіх введених до цього літер із літерами слова. Помилкове ж натискання сусідньої клавіші під час введення слова одразу ж виключить необхідне слово зі списку пропозицій, оскільки це слово не відповідатиме вже введеній послідовності літер, що в свою чергу, призведе до невірного результату роботи методу.

Тому, пропонується модифікація даного методу, яка полягає у використанні блоків літер, згрупованих відповідно до їх розташування на клавіатурі. Найбільш поширеною клавіатурою, яка встановлюється на сучасних мобільних пристроях, є клавіатура, що зазвичай застосовується при роботі із настільними комп'ютерами та ноутбуками, або так звана QWERTY-клавіатура, на якій кожній літері відповідає певна клавіша. На мобільних пристроях із сенсорним дисплеєм така клавіатура розміщується на екрані пристрою і натискання клавіш на ній відбувається шляхом натискання у відповідній області екрану. Враховуючи це, розглядатимемо модифікацію методу предиктивного введення тексту на основі *N*-грам, що пропонується, у контексті введення тексту за допомогою саме такої клавіатури. Однак, слід зазначити, що дана модифікація також може бути застосовною і для деяких інших клавіатур, в тому числі і 12-кнопкових

клавіатур, які встановлювались на переважній більшості мобільних телефонів кінця 1990-х років – початку 2000-х років.

Використання блоків літер, згрупованих відповідно до розташування цих літер на клавіатурі, сприятиме коректній роботі методу предиктивного введення тексту на основі *N*-грам навіть при помилковому натисканні сусідньої літери при введенні деякого слова, що при швидкому введенні тексту є достатньо ймовірним.

Отже, згідно модифікації, що пропонується, при натисканні деякої клавіші на клавіатурі відбувається вибір не окремої літери, яка відповідає даній клавіші, а певної групи літер, відповідно до поточного розподілу літер між блоками.

Розподіл літер між блоками, що не перетинаються

Одним із можливих варіантів розподілу літер між блоками є розподіл літер таким чином, щоб кожній літері відповідав один блок [24]. Іншими словами, в даному випадку відбувається поділ клавіатури на окремі блоки літер, що не перетинаються між собою. Наприклад, набір таких блоків літер може складатися з шести елементів відповідно до розташування цих літер на клавіатурі, причому кожному з трьох рядків літер на QWERTY-клавіатурі відповідає два блоки – лівий та правий. Такий приклад розподілу літер між блоками для американського варіанту QWERTY-клавіатури наведено на рис. 9 [24].

Як видно, в даному випадку літери на клавіатурі групуються наступним чином: літери «Q», «W», «E», «R» та «T» розташовуються у першому блоці, літери «Y», «U», «I», «O» та «P» – у другому, літери «A», «S», «D», «F» та «G» – у третьому, літери «H», «J», «K» та «L» – у четвертому, літери «Z», «X», «C» та «V» – у п’ятому, і літери «B», «N» та «M» – у шостому.

Відповідно, при натисканні деякої клавіші відбувається вибір не окремої літери, що відповідає цій клавіші, а блоку літер загалом, в якому міститься дана літера. Іншими словами, при здійсненні після натискання

деякої клавіші прогнозування слова, що користувач має намір ввести, розглядається не тільки літера, що відповідає цій клавіші, а й інші літери з блоку літер, в якому міститься дана літера. Так, наприклад, після натискання користувачем клавіші із позначенням літери «N» при здійсненні прогнозування слова розглядаються літери із блоку №6 – «B», «N» та «M».

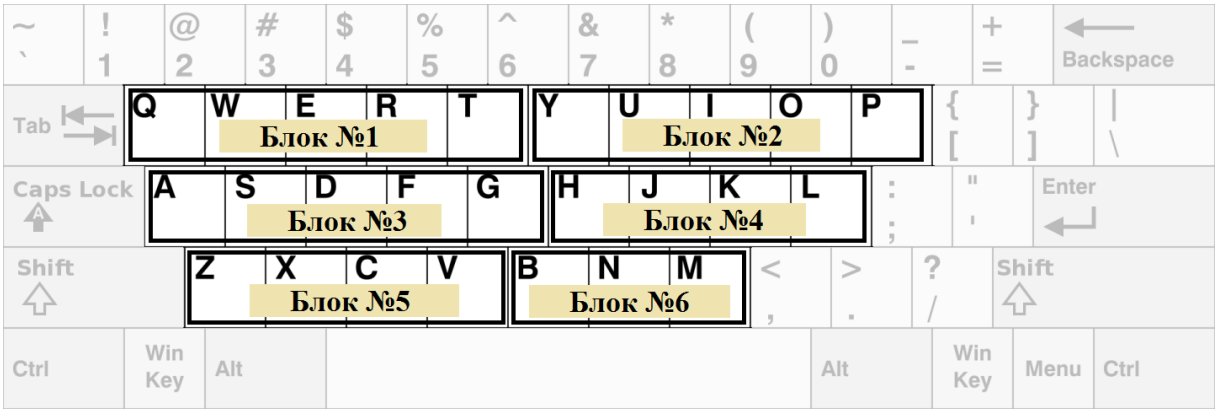


Рис. 9. Приклад розподілу літер між окремими блоками для американського варіанту QWERTY-клавіатури

Слід зазначити, що розподіл літер між блоками може варіюватись в залежності від мови, на якій здійснюється введення. Так, наприклад, при введенні тексту українською чи російською мовами застосовується більша кількість клавіш із позначенням літер, ніж при введенні тексту англійською мовою. Це пов’язано з тим, що кількість літер в українському чи російському алфавіті є більшою за кількість літер в англійському алфавіті, але на QWERTY-клавітурах, що розглядаються, кожній літері алфавіту відповідає певна клавіша. Таким чином, розподіл літер між блоками для різних варіантів QWERTY-клавіатур може бути різним. На рис. 10 наведено приклад розподілу літер між блоками для українського варіанту QWERTY-клавіатури, а на рис. 11 – для російського.

Рис. 10. Приклад розподілу літер між окремими блоками для українського варіанту QWERTY-клавіатури

Рис. 11. Приклад розподілу літер між окремими блоками для російського варіанту QWERTY-клавіатури

Як видно, для українського та російського варіанту QWERTY-клавіатури в даному випадку розподіл літер між блоками літер відбувається шляхом використання не шести блоків, а восьми, оскільки загальна кількість клавіш на клавіатурі, що відповідають літерам, є більшою. Подібним чином може здійснюватися розподіл літер між блоками літер і для клавіатур, що використовуються для введення тексту на інших мовах.

Однак, при використанні підходу, що передбачає розподіл літер між блоками таким чином, що кожна літера міститься лише в одному блоці, можлива некоректна робота методу предиктивного введення тексту в тому разі, якщо відбувається помилкове натискання користувачем літери, що

знаходиться поруч із літерою, яку користувач мав намір ввести, однак належить до сусіднього блоку. В такому випадку при здійсненні прогнозування слова розглядатиметься набір літер, що не міститиме бажаної літери. Це, в свою чергу, може призвести до того, що у списку пропозицій для введення слово, яке мав намір ввести користувач, буде відсутнім, а загальна швидкість введення, відповідно, буде нижчою.

Так, наприклад, у випадку, коли користувач має намір ввести англійську літеру «G», можливим є як помилкове натискання літери «F», що розташована ліворуч від літери «G», так і літери «H», що розташована праворуч від літери «G». Однак, тоді як літера «F» входить до того самого блоку, що й літера «G», літера «H» до цього блоку не входить. Відповідно, при помилковому натисканні літери «F» у цьому випадку бажана літера «G» увійде до набору літер, що розглядатимуться при здійсненні прогнозування слова, а при помилковому натисканні літери «H» – не увійде.

Розглядаючи можливі помилкові натискання літер, що розташовані ліворуч або праворуч від бажаних, на наведеному вище прикладі розподілу літер між шістьма блоками при використанні американського варіанту QWERTY-клавіатури, видно, що на межі блоків розташовані шість літер – «T», «Y», «G», «H», «V» та «B». Відповідно, кількість можливих варіантів помилкових натискань літер, що розташовані ліворуч або праворуч від бажаних, при яких бажана літера не потрапить до набору літер, що розглядатимуться при здійсненні прогнозування слова, в даному випадку також дорівнюватиме шести – кількості літер, що розташовані на межі блоків.

Загальна кількість можливих варіантів помилкових натискань літер, що розташовані ліворуч або праворуч від бажаних, становить $\sum N_{LR}(l)$, де $N_{LR}(l)$ – кількість літер, що розташовані ліворуч або праворуч від деякої літери l. У випадку американського варіанту QWERTY-клавіатури 3 літери – «Q», «A» та «Z» не мають сусідніх літер ліворуч, 3 літери – «P», «L» та

«M» не мають сусідніх літер праворуч, а інші 20 літер мають сусідні літери як ліворуч, так і праворуч. Відповідно, загальна кількість можливих варіантів помилкових натискань літер, що розташовані ліворуч або праворуч від бажаних, в даному випадку становитиме:

$$3 * 1 + 3 * 1 + 20 * 2 = 46.$$

Таким чином, припустивши, що помилкові натискання літер, що розташовані ліворуч та праворуч від бажаної, є такими, що мають однакову ймовірність, а також те, що ймовірність введення кожної з літер на клавіатурі є однаковою, ймовірність того, що при помилковому натисканні клавіш, що розташовані ліворуч або праворуч від бажаних, бажана літера не потрапить до набору літер, що розглядатимуться при здійсненні прогнозування слова, становитиме:

$$\frac{N_B}{\sum N_{LR}(l)} = \frac{6}{46} = 0{,}13,$$

де N_B – кількість літер, що розташовані на межі блоків.

Отже, у 13% випадків помилкових натискань літер, що розташовані ліворуч або праворуч від бажаних, бажана літера не потрапить до набору літер, що розглядатимуться при здійсненні прогнозування слова.

Однак, слід зазначити, що окрім помилкових натискань літер, що розташовані ліворуч або праворуч від бажаних, можуть також відбуватися помилкові натискання літер, що розташовані вище або нижче від бажаних. Відповідно, у випадку наведеного вище розподілу літер між шістьма блоками будь-яке помилкове натискання літери, що розташована вище або нижче від бажаної, призведе до того, що бажана літера не потрапить до набору літер, що розглядатимуться при здійсненні прогнозування слова, оскільки літери, які розташовуються вище або нижче бажаної, не містяться разом з нею у одному блоці.

Наприклад, у випадку, коли користувач має намір ввести англійську літеру «G», окрім можливих помилкових натискань літер «F» або «H», що розташовані відповідно ліворуч та праворуч від неї, можливими є також як

помилкові натискання літер «T» або «Y», що розташовані вище від літери «G», так і літер «V» або «B», що розташовані нижче від літери «G». Однак, жодна з літер «T», «Y», «V» або «B» не входить до того самого блоку, що й літера «G».

Загальна кількість можливих варіантів помилкових натискань літер, що розташовані ліворуч, праворуч, вище або нижче від бажаних, становить $\sum N_{LRUD}(l)$, де $N_{LRUD}(l)$ – кількість літер, що розташовані ліворуч, праворуч, вище або нижче від деякої літери l. В свою чергу,

$$N_{LRUD}(l) = N_{LR}(l) + N_{UD}(l),$$

де $N_{UD}(l)$ – кількість літер, що розташовані вище або нижче від деякої літери l.

У випадку американського варіанту QWERTY-клавіатури літери, що розташовані у першому рядку, не мають сусідніх літер вище, літери, що розташовані у третьому рядку – нижче, а літери, що розташовані у другому рядку, можуть мати сусідні літери як вище, так і нижче.

У першому рядку кожна з 2-х літер «Q» та «P» має одну сусідню літеру нижче, а кожна з інших 8-ми літер має дві сусідні літери нижче. У другому рядку кожна з 9-ти літер має дві сусідні літери вище. Також, у другому рядку кожна з 2-х літер «A» та «K» має одну сусідню літеру нижче, літера «L» не має сусідніх літер нижче, а кожна з інших 6-ти літер має дві сусідні літери нижче. У третьому рядку кожна з 7-ми літер має дві сусідні літери вище.

Таким чином, для першого рядка кількість можливих варіантів помилкових натискань літер, що розташовані вище або нижче від бажаних, становитиме:

$$2 * 1 + 8 * 2 = 18.$$

Для другого рядка кількість можливих варіантів помилкових натискань літер, що розташовані вище або нижче від бажаних, становитиме:

$$9 * 2 + 2 * 1 + 6 * 2 = 32.$$

В свою чергу, для третього рядка кількість можливих варіантів помилкових натискань літер, що розташовані вище або нижче від бажаних, становитиме:

$$7 * 2 = 14.$$

Відповідно, загальна кількість можливих варіантів помилкових натискань літер, що розташовані вище або нижче від бажаних, в даному випадку становитиме:

$$18 + 32 + 14 = 64.$$

Оскільки $\sum N_{LRUD}(l) = \sum N_{LR}(l) + \sum N_{UD}(l)$, то загальна кількість можливих варіантів помилкових натискань літер, що розташовані ліворуч, праворуч, вище або нижче від бажаних, становитиме:

$$46 + 64 = 110.$$

Таким чином, припустивши, що помилкові натискання літер, що розташовані ліворуч, праворуч, вище або нижче від бажаної, є такими, що мають однакову ймовірність, а також те, що ймовірність введення кожної з літер на клавіатурі є однаковою, ймовірність того, що при помилковому натисканні клавіш, що розташовані ліворуч, праворуч, вище або нижче від бажаних, бажана літера не потрапить до набору літер, що розглядатимуться при здійсненні прогнозування слова, становитиме:

$$\frac{\sum N_{LRUD}(l) - (\sum N_{LR}(l) - N_B)}{\sum N_{LRUD}(l)} = \frac{110 - (46 - 6)}{110} = 0{,}64.$$

Отже, відсоток випадків помилкових натискань літер, що розташовані ліворуч, праворуч, вище або нижче від бажаних, при яких бажана літера не потрапить до набору літер, що розглядатимуться при здійсненні прогнозування слова, становитиме вже 64%, на відміну від 13%, отриманих при розгляді помилкових натискань літер, що розташовані лише ліворуч або праворуч від бажаних.

Враховуючи те, що при застосуванні даного розподілу літер між блоками у більше ніж половині випадків помилкових натискань сусідніх літер бажана літера не потраплятиме до набору літер, що розглядатимуться

при здійсненні прогнозування слова, необхідним є застосування розподілу літер між блоками таким чином, щоб враховувалися як можливі помилкові натискання літер, що розташовані ліворуч або праворуч від бажаної літери, так і можливі помилкові натискання літер, що розташовані вище або нижче від неї. Причому, ще більшої ефективності можна досягнути у тому випадку, якщо дві сусідні літери завжди будуть міститися в одному і тому самому блоці.

Розподіл літер між блоками, що перетинаються

Для цього розподілимо клавіатуру на блоки таким чином, що кожна літера утворюватиме свій окремий блок, який складатиметься із цієї літери, а також із літер, що розташовуються ліворуч, праворуч, вище або нижче від неї.

Таким чином, у цьому випадку блоки перетинатимуться між собою, а кожна літера, відповідно, буде міститися у декількох блоках одночасно. Це сприятиме тому, що при помилковому натисканні літер, що розташовані як ліворуч або праворуч, так і вище або нижче від бажаних, бажана літера гарантовано потраплятиме до набору літер, що розглядатимуться при здійсненні прогнозування слова.

Так, наприклад, у блоці, що утворює літера «G», розташовуватиметься літера «F», що розташована ліворуч від літери «G», літера «H», що розташована праворуч від літери «G», літери «T» та «Y», що розташовані вище від літери «G», літери «V» та «B», що розташовані нижче від літери «G», а також безпосередньо літера «G». Приклад створення блоку, що утворює літера «G», наведено на рис. 12.

Відповідно, до складу блоків, які утворюють літери, що не мають сусідніх літер з деякого боку, входитимуть лише ті літери, що є сусідніми з нею з інших боків. Наприклад у блоці, що утворює літера «A», розташовуватиметься літера «S», що розташована праворуч від літери «A», літери «Q» та «W», що розташовані вище від літери «A», літера «Z», що

розташована нижче від літери «A», а також безпосередньо літера «A». Приклад створення блоку, що утворює літера «A», наведено на рис. 13.

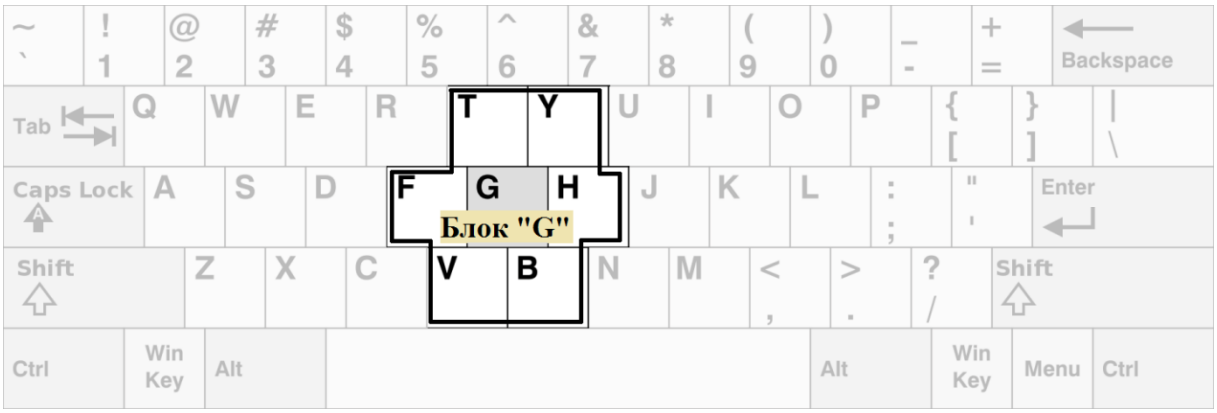


Рис. 12. Приклад створення блоку, що утворює літера «G», для американського варіанту QWERTY-клавіатури

Рис. 13. Приклад створення блоку, що утворює літера «A», для американського варіанту QWERTY-клавіатури

Аналогічним чином пропонується створення й блоків, що утворюють інші літери. Оскільки кожна літера утворюватиме свій окремий блок, загальна кількість блоків, на які буде розподілено літери на клавіатурі, дорівнюватиме кількості цих літер. Кількість літер у блоках для деяких літер може бути однаковою, а для деяких – різною. Так, наприклад, блок, що утворює літера «Q» складатиметься всього з трьох літер – «Q», «W» та «A», оскільки літера «Q» не має сусідніх літер ліворуч та вище від себе, а має лише по одній сусідній літері праворуч та нижче від себе – літери «W»

та «A» відповідно. В свою чергу, блок, що утворює літера «D», міститиме вже сім літер, оскільки, окрім безпосередньо літери «D», до цього блоку входитимуть також літера «S», що розташована ліворуч від неї, літера «F», що розташована праворуч, літери «E» та «R», що розташовані вище, а також літери «X» та «C», що розташовані нижче.

2.3.3. Модифікація методу предиктивного введення тексту на мобільних пристроях на основі N-грам, що пропонується

Отже, запропонований модифікований метод предиктивного введення тексту на мобільних пристроях на основі N-грам можна сформулювати таким чином.

На основі введеної користувачем послідовності літер формується послідовність блоків таким чином, що до кожного блоку входить літера, яку було натиснуто, а також сусідні літери – літери, що розташовуються ліворуч, праворуч, вище та нижче від неї.

Після цього серед слів, дані про частоти уніграм яких наявні у програмі, відбувається пошук слів, що відповідають заданій послідовності блоків.

Отриманий набір слів сортується згідно з частотами N-грам, до складу яких входять попередньо введені користувачем слова та слово з отриманого набору, причому триграми мають найвищий пріоритет, а уніграми – найнижчий.

Перші P слів з відсортованого набору слів, де P – максимальна кількість пропозицій для введення, відображаються користувачеві. При виборі користувачем слова зі списку це слово підставляється у текст, що вводиться, замінюючи введену послідовність літер.

3. ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ПРОГРАМНИХ ЗАСОБІВ

3.1. Вимоги до розроблюваних програмних засобів

Для реалізації і тестування модифікованого методу предиктивного введення тексту на мобільних пристроях на основі N-грам необхідним є створення програмних засобів, що забезпечуватимуть можливість введення тексту користувачем і відображення користувачеві списку пропозицій для введення відповідно до модифікації методу предиктивного введення тексту на основі N-грам, що пропонується.

Для взаємодії користувача із програмними засобами в даному випадку достатнім є застосування інтерфейсу користувача у вигляді консольного додатку.

Також, слід забезпечити можливість врахування слів, що вводить користувач, при подальших створеннях списків пропозицій для введення з метою пристосування програмних засобів до специфіки введення тексту тим чи іншим користувачем.

Таким чином, необхідним є створення програмних засобів, що забезпечуватимуть наступну функціональність:

1) здійснення генерування N-грам слів та підрахунок частот цих N-грам на основі деякого вхідного текстового корпусу;
2) створення інтерфейсу користувача у вигляді консольного додатку та забезпечення можливості здійснення введення тексту користувачем;
3) створення списку пропозицій для введення під час введення тексту користувачем та відображення цього списку користувачеві із можливістю вибору слова зі списку або продовження введення тексту без застосування пропозицій;
4) врахування слів, що вводить користувач, при створенні наступних списків пропозицій для введення.

3.2. Обґрунтування вибору мови програмування

Сьогодні існує велика кількість різноманітних мов програмування, причому вирішення однієї і тієї самої задачі може здійснюватися за допомогою програм, написаних на різних мовах. Крім того, деякі мови краще застосовувати для вирішення одних задач, а деякі – для вирішення інших. Для реалізації програмних засобів, що надаватимуть можливість здійснювати генерування *N*-грам слів та підрахунок їх частот на основі деякого текстового корпусу, дозволятимуть користувачеві здійснювати введення тексту та пропонуватимуть користувачеві під час введення слова для введення, а також враховуватимуть слова, які вводить користувач, при створенні наступних списків пропозицій для введення, доцільним є застосування мови програмування, що є простою у використанні, дозволяє швидко створювати необхідні компоненти програмних засобів, і не потребує написання великого об’єму коду для їх реалізації. Також, для проведення тестування роботи програмних засобів за допомогою введення тексту і створення списку пропозицій для введення у реальному часі доцільним є застосування мови програмування, що надаватиме можливість здійснювати виконання необхідних обчислень за прийнятний час.

3.2.1. Мова програмування Python

Python – високорівнева мова програмування загального призначення, орієнтована на підвищення продуктивності розроблення програмного забезпечення і спрощення читання коду [25]. Синтаксис ядра мови програмування Python є мінімалістичним, однак, у той же час стандартна бібліотека мови Python містить достатньо великий обсяг корисних функцій.

Мова програмування Python підтримує структурне, об’єктно-орієнтоване, функціональне, імперативне і аспектно-орієнтоване програмування. Основними архітектурними рисами мови Python є динамічна типізація, автоматичне керування пам’яттю, повна інтроспекція, механізм оброблення виключень, підтримка багатопоточних (англ. «multi-

threading») обчислень, високорівневі структури даних. Підтримується також розбиття програм на модулі, які, в свою чергу, можуть об’єднуватися в пакети.

Мова програмування Python працює майже на всіх відомих платформах – Microsoft Windows, FreeBSD, Linux, macOS, iOS, Windows Mobile, Symbian, Android та інших.

Розглядаючи типи та структури даних мови програмування Python, слід зазначити, що мова Python підтримує динамічну типізацію, тобто тип змінної визначається лише під час виконання. Тому замість так званого присвоювання значення змінній слід говорити про так зване зв’язування значення з деяким ім’ям. В Python є такі вбудовані типи даних, як булевий, рядок, Unicode-рядок, ціле число, число з плаваючою комою, комплексне число і деякі інші. Також в Python вбудовані такі колекції, як список, кортеж (незмінний список), словник, множина та інші. Всі значення у мові програмування Python є об’єктами, в тому числі функції, методи, модулі, класи.

Мова програмування Python має чіткий і послідовний синтаксис, продуману модульність та масштабованість, завдяки чому читання коду програм, написаних на Python, є достатньо легким.

Стандартна бібліотека є однією з привабливих сторін мови програмування Python. Вона містить засоби для роботи з багатьма мережевими протоколами і форматами Інтернету, наприклад, модулі для написання HTTP-серверів і клієнтів, для розбору і створення поштових повідомлень, для роботи з XML тощо. Набір модулів для роботи з операційною системою дозволяє розробляти кросплатформні програмні засоби. Існують також модулі для роботи з регулярними виразами, текстовими кодуваннями, мультимедійними форматами, криптографічними протоколами, архівами, модулі серіалізації даних, підтримка юніт-тестування та інше.

Python є стабільною і поширеною мовою програмування. Вона використовується в багатьох проектах і в різних якостях: як основна мова програмування або для створення розширень і інтеграції додатків. На мові Python реалізовано велику кількість проектів, також вона активно використовується для створення прототипів майбутніх програм. Мова Python використовується в багатьох великих компаніях: Dropbox, Google (наприклад деякі частини Youtube і Youtube API написані на Python), Facebook, Instagram.

3.2.2. Мова програмування Java

Java – сильно типізована об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems, що в подальшому була придбана компанією Oracle [26]. Програми Java зазвичай транслюються в спеціальний байт-код, тому вони можуть працювати на будь-якій комп'ютерній архітектурі за допомогою віртуальної Java-машини. Датою офіційного випуску є 23 травня 1995 року. Станом на 2019 рік Java є однією з найпопулярніших мов програмування.

Програми, написані на мові програмування Java, транслюються в байт-код Java, який виконується віртуальною машиною Java (англ. «JVM» – «Java Virtual Machine») – програмою, що оброблює байтовий код і передає інструкції обладнанню у якості інтерпретатора.

Перевагою такого способу виконання програм є повна незалежність байт-коду від операційної системи і обладнання, що, в свою чергу, дозволяє виконувати програми, написані на мові програмування Java, на будь-якому пристрої, для якого існує відповідна віртуальна машина. Іншою важливою особливістю технології Java є гнучка система безпеки, в рамках якої виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують встановлені повноваження програми, наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером, викликають негайне переривання.

Часто до недоліків концепції віртуальної машини відносять зниження продуктивності. Однак, завдяки проведенню ряду удосконалень швидкість виконання програм на Java було дещо збільшено. До таких удосконалень можна віднести застосування технології трансляції байт-коду в машинний код безпосередньо під час роботи програми (JIT-технологія) з можливістю збереження версій класу в машинному коді, широке використання платформно-орієнтованого коду, або так званого native-коду, в стандартних бібліотеках, використання апаратних засобів, що забезпечують прискорене оброблення байт-коду, наприклад, технологія Jazelle, що підтримується деякими процесорами архітектури ARM.

За даними досліджень, проведених сайтом shootout.alioth.debian.org, для семи різних задач час виконання програми на мові програмування Java становить в середньому є в півтора-два рази більшим, ніж для мов програмування C/C++, в деяких випадках Java швидше, а в окремих випадках в 7 разів повільніше. З іншого боку, для більшості з них споживання пам’яті Java-машиною було в 10-30 разів більшим, ніж програмою, написаною на C/C++.

Програми, написані на мові програмування Java, мають репутацію більш повільних і таких, що займають більше оперативної пам’яті, ніж програми, написані на мові C. Проте, швидкість виконання програм, написаних на мові Java, була істотно поліпшена із випуском в 1997-1998 роках так званого JIT-компілятора в версії 1.1 у якості доповнення до інших особливостей мови для підтримки кращого аналізу коду.

Крім того, було проведено оптимізацію віртуальної машини Java – з 2000 року для цього використовується віртуальна машина HotSpot. Станом на лютий 2012 року, код Java 7 є приблизно в 1,8 рази повільнішим за код, написаний на мові C.

Із залученням Java-технологій було реалізовано успішні проекти, серед яких можна виділити RuneScape, Amazon, eBay, LinkedIn, Yahoo! та інші.

3.2.3. Мова програмування Perl

Perl – високорівнева інтерпретована динамічна мова програмування загального призначення, створена американським програмістом Ларрі Уоллом (англ. Larry Wall), лінгвістом за освітою [27]. Назва мови офіційно розшифровується як «Practical Extraction and Report Language» (англ. – практична мова для отримання даних та складання звітів).

Основною особливістю мови програмування Perl вважаються її багаті можливості для роботи з текстом, в тому числі робота з регулярними виразами, вбудована в синтаксис. Мова Perl успадкувала багато властивостей від мов C, AWK, скриптових мов командних оболонок UNIX.

Perl – мова програмування загального призначення, яку було спочатку створено для маніпуляцій з текстом, однак на даний момент вона використовується для виконання широкого спектру задач, включаючи системне адміністрування, розроблення веб-застосунків, мережеве програмування, ігри, біоінформатику, розроблення графічних інтерфейсів користувача.

Загальна структура мови програмування Perl в загальних рисах веде свій початок від мови C. Мова Perl є процедурною за своєю природою, має змінні, вирази присвоювання, блоки коду, що відокремлюються фігурними дужками, керуючі структури і функції.

Мова Perl також запозичує ряд властивостей з мов програмування командних оболонок UNIX. Всі змінні маркуються провідними знаками, які точно виражають тип даних змінної в цьому контексті (наприклад, скаляр, масив, хеш тощо). Важливим є те, що ці знаки дозволяють змінним бути інтерпольованими в рядках. Мова Perl має велику кількість вбудованих функцій, які забезпечують інструментарій, що часто використовується для програмування оболонки, наприклад, сортування або виклик системних служб.

Всі версії мови програмування Perl виконують автоматичну типізацію даних і автоматичний контроль над пам’яттю. Інтерпретатор має

відомості про тип і запити пам’яті кожного об’єкта програми, він здійснює розподілення і звільнення пам’яті, виконуючи при цьому підрахунок посилань. Переведення одного типу даних в інший, наприклад, числа в рядок, відбувається автоматично під час виконання, переведення ж типів даних, що є неможливими для виконання, призводять до фатальної помилки.

Користувачі операційної системи Microsoft Windows зазвичай використовують дистрибутиви з поередньо скомпільованими бінарними файлами, такі, як ActivePerl або Strawberry Perl, оскільки компіляція мови Perl з кодів програми в цій операційній системі є не найпростішим завдання, яке, однак, може бути полегшеним завдяки використанню Cygwin – UNIX-подібного середовища та інтерфейсу командного рядка для операційної системи Microsoft Windows.

3.2.4. Порівняння мов програмування

Передбачається, що програми, написані на мові програмування Python, в цілому працюють повільніше, ніж програми, написані на мові Java, але при цьому їх розроблення займає набагато менше часу [28].

Програми, написані на мові Python, зазвичай є в 3-5 разів коротшими, ніж їх аналоги, написані на мові Java. Ця різниця може бути приписана типам даних і їх динамічної типізації, що вбудовані в Python на високому рівні.

Наприклад, програміст, що пише на мові програмування Python, не витрачає час на оголошення типів аргументів або змінних, а численні види поліморфних списків і словників, синтаксична підтримка яких вбудована в саму мову, знаходять застосування практично в кожній програмі, написаній на мові Python.

Через динамічну типізацію виконання програми, написаної на мові Python, здійснюється важче, ніж у випадку з мовою Java. Наприклад, обчислюючи вираз $a+b$, програма повинна спочатку визначити тип об’єктів a і b, адже при компіляції він є невідомим. Потім програма здійснює запит

відповідної операції додавання, яка може бути перевантажена методом, визначеним користувачем.

В мові Java, в свою чергу, може здійснюватися ефективне додавання цілих чисел або чисел з плаваючою комою, але необхідним є декларування змінних *a* і *b*, а також не дозволяється перевантаження оператора «+», наприклад, класами, визначеними користувачем.

Виходячи з цього, мова Python набагато краще підходить в якості інтегруючої мови, в той час як мову Java можна скоріше охарактеризувати як мову низькорівневого виконання.

В свою чергу, обидві мови програмування Python та Perl зазнали впливу скриптів UNIX. Вони мають багато схожих рис, однак різну філософію.

В мові програмування Perl здійснюється акцент на підтримці загальних задач з орієнтацією на програму. Наприклад, в мові Perl є вбудована підтримка регулярних виразів і функціональність для сканування файлів і генерування звітів. В свою чергу, в мові Python більше підтримуються загальні методології, наприклад, структурування даних та об’єктно-орієнтоване програмування.

Надаючи програмістам елегантну і не надмірно заплутану систему позначень, мова програмування Python спонукає їх писати код, легкий для читання, і, таким чином, також і легкий для обслуговування.

Незважаючи на близькість цих двох мов, вони не конкурують одне з одним, оскільки мова Perl має явну перевагу в додатках, а мова Python знаходить застосування далеко за межами ніші мови Perl.

Таким чином, враховуючи те, що процес розроблення програмних засобів на мові програмування Python є простішим, а програмний код, написаний на цій мові є більш легким для читання, ніж на мові програмування Perl, а також те, що процес розроблення програмних засобів на мові програмування Python потребує менших обсягів програмного коду, ніж на мові Java, для розроблення програмних засобів,

що надаватимуть можливість здійснювати генерування *N*-грам слів та підрахунок їх частот на основі деякого текстового корпусу, дозволятимуть користувачеві здійснювати введення тексту та пропонуватимуть користувачеві під час введення слова для введення, а також враховуватимуть слова, які вводить користувач, при створенні наступних списків пропозицій для введення, було обрано мову програмування Python.

3.3. Структурна організація програмних засобів

Програмні засоби розроблено на мові програмування Python версії 2.7.11 з використанням середовища розробки PyCharm Community Edition версії 3.4.3 на операційній системі Windows. Слід зазначити, що при створенні програмних засобів не було використано сторонніх бібліотек, усі операції виконуються за допомогою використання стандартної бібліотеки мови програмування Python.

Для здійснення операцій введення та виведення у модулі взаємодії з користувачем використано в тому числі модуль msvcrt зі стандартної бібліотеки мови програмування Python, що при розробленні програмних засобів був призначений для взаємодії із клавіатурою при роботі на операційній системі Windows, а саме, використовується метод getch(). Використання модуля msvcrt дозволило спростити розроблення даних програмних засобів, оскільки забезпечення кросплатформності модуля взаємодії з користувачем потребувало б налаштування також і інших модулів, призначених для взаємодії з користувачем при роботі на UNIX-подібних операційних системах.

Однак, слід зазначити, що модуль msvcrt використовується лише у модулі взаємодії з користувачем, і тому, відповідно, робота інших модулів розроблених програмних засобів є незалежною від операційної системи, що використовується. Тому для забезпечення підтримки роботи на інших операційних системах всіх програмних засобів в цілому необхідним є лише вдосконалення модуля взаємодії з користувачем шляхом забезпечення

використання інших модулів зі стандартної бібліотеки мови програмування Python у випадку виконання програмних засобів на інших операційних системах. Інші компоненти, до яких, в тому числі, відносяться модулі, що реалізовують безпосередньо генерування списку пропозицій для введення на основі тексту, який вводить користувач, що є основою розроблених програмних засобів, таких вдосконалень не потребують і можуть працювати також і на інших операційних системах.

Розроблені програмні засоби складаються з наступних модулів:

1) n_grams_generator.py – модуль генерування N-грам слів та підрахунку їх частот на основі деякого текстового корпусу;
2) n_grams_manager.py – модуль зчитування частот N-грам слів з файлів та запису їх у файли;
3) propositions_creator.py – модуль, що виступає у ролі обгортки над модулем генерування списку пропозицій для введення, здійснюючи попереднє оброблення тексту, введеного користувачем, а також виклик відповідних методів з модуля генерування списку пропозицій для введення в залежності від тексту, поданого на вхід даного модуля;
4) specific_propositions_creator.py – модуль генерування списку пропозицій для введення;
5) candidates_creator.py – модуль генерування можливих початків слів в залежності від розподілу літер між блоками;
6) n_grams_updater.py – модуль, що здійснює оновлення частот N-грам слів на основі тексту, який вводить користувач;
7) main.py – модуль інтерфейсу взаємодії із користувачем.

Також розроблені програмні засоби містять такі допоміжні текстові файли:

1) letter_blocks.txt – файл, у якому міститься інформація про розподіл літер між блоками;

2) n_grams/n_grams_1.txt, n_grams/n_grams_2.txt та n_grams/n_grams_3.txt – файли, у які записано дані про частоти відповідно уніграм, біграм та триграм слів, отримані в процесі генерування цих *N*-грам слів на основі деякого текстового корпусу;
3) n_grams_user/n_grams_1.txt, n_grams_user/n_grams_2.txt та n_grams_user/n_grams_3.txt – файли, у які записано дані про частоти відповідно уніграм, біграм та триграм слів, отримані безпосередньо в процесі введення тексту користувачем.

Крім того, розроблені програмні засоби містять директорію texts, в якій за замовчуванням відбувається пошук файлів, що утворюють текстовий корпус.

Загальну структуру розроблених програмних засобів наведено на рис. 14.

3.3.1. Модуль генерування N-грам слів та підрахунку їх частот на основі деякого текстового корпусу

Модуль генерування *N*-грам слів та підрахунку їх частот на основі деякого текстового корпусу отримує на вхід заданий текстовий корпус і здійснює на його основі генерування уніграм, біграм та триграм слів, а також підрахунок їх частот, після чого передає дані у модуль зчитування частот *N*-грам слів з файлів та запису їх у файли для запису отриманих частот уніграм, біграм та триграм у допоміжні файли n_grams/n_grams_1.txt, n_grams/n_grams_2.txt та n_grams/n_grams_3.txt.

Алгоритм роботи модуля генерування *N*-грам слів та підрахунку їх частот на основі деякого текстового корпусу виглядає наступним чином:

1. Отримання на вхід назви директорії, у якій розташовується необхідний текстовий корпус.
2. Отримання з вказаного текстового корпусу набору токенів шляхом зчитування даних почергово з кожного файлу, що міститься у вказаній директорії, і розбиття на токени зчитаного

тексту відповідно з кожного із файлів. Слід зазначити, що в даному випадку для спрощення реалізації програмних засобів розбиття на токени відбувається з урахуванням того, що слова вважаються такими, що до переліку символів, з яких складаються слова, входять тільки літери латинського алфавіту та апостроф.

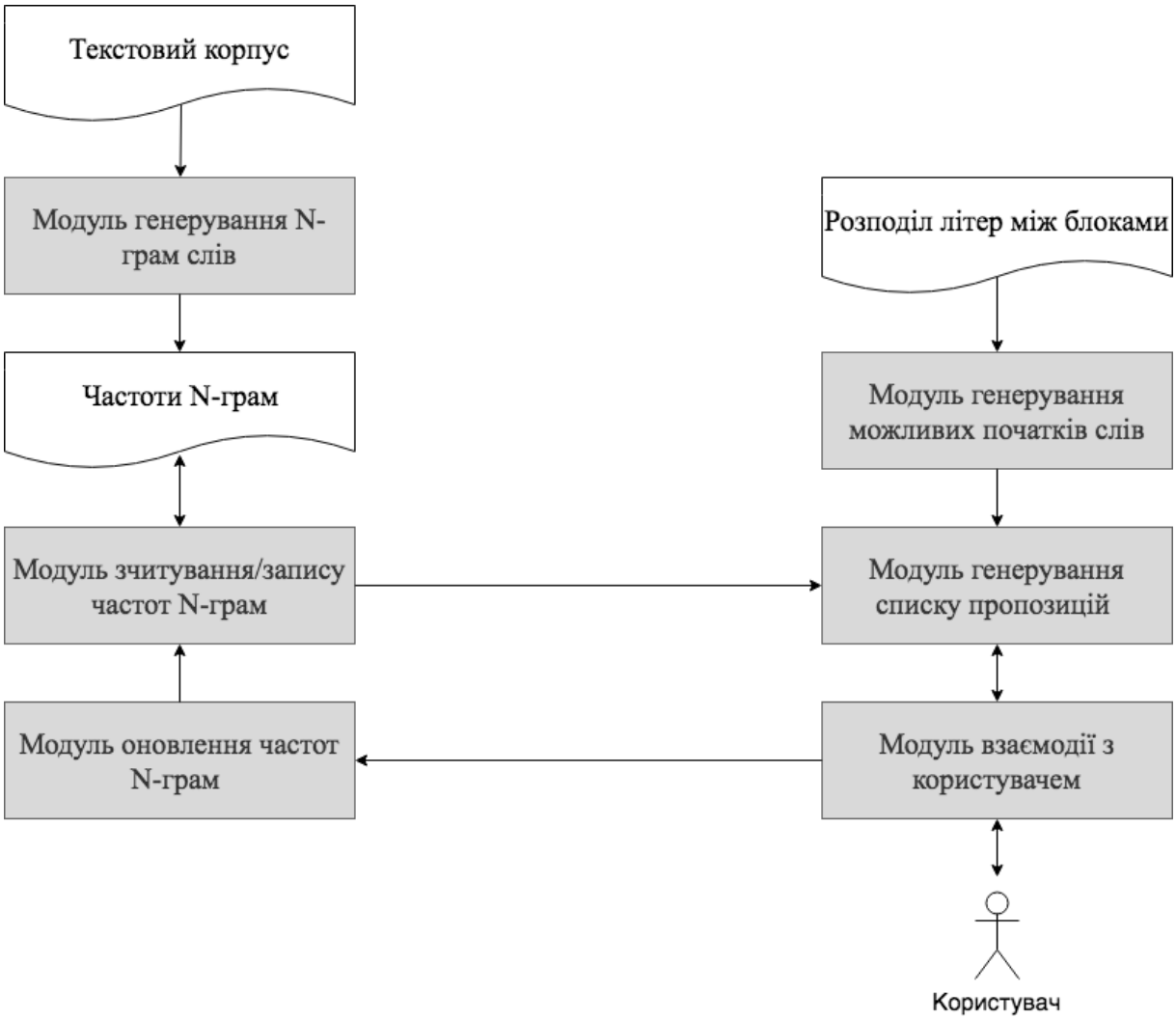


Рис. 14. Загальна структура розроблених програмних засобів

3. З отриманого набору токенів виокремлюється список слів, на основі якого генерується набір уніграм та відбувається підрахунок їх частот.

4. На основі отриманого набору токенів генерується набір біграм і триграм та відбувається підрахунок їх частот.
5. Відбувається здійснення виклику відповідного методу з модуля зчитування частот N-грам слів з файлів та запису їх у файли для запису отриманих уніграм, біграм і триграм та їх частот у відповідні файли.

Слід зазначити, що при здійсненні генерування уніграм, біграм та триграм слів на основі текстового корпусу, поданого на вхід, відбувається врахування спеціальним чином регістру слів, що підлягають обробленню. Так, оскільки в тексті більшість слів можуть починатися як з малої літери, так і з великої літери, запис у файл відбувається саме слова, написаного з малої літери, а частоти, отримані для того ж самого слова, але такого, що починається з великої літери, об'єднуються із частотами слова, що починається з малої літери. Це пов'язано з тим, що такі слова є однаковими, однак збереження у файл окремо обох варіантів написання слова призвело до б того, що одне й те саме слово могло б враховуватися двічі, що потенційно могло б призвести до менш ефективної роботи методу предиктивного тексту. Однак, слід зазначити, що в тексті також можуть зустрічатися і слова, які мають тільки варіант написання з великої літери, наприклад, англійське слово «I» (англ. – я), включаючи також і можливі скорочення після нього відповідно слів «am», «have» або «had» («I'm», «I`ve», «I'd»), або ж власні назви імен, географічних об'єктів тощо («Elizabeth», «Washington» та інші.), хоча й деякі з них можуть зустрічатися як у якості власної, так і загальної назви. Також, у тексті можуть зустрічатися абревіатури, тобто слова, що складаються лише з великих літер, наприклад, «NASA», «USA» та інші. Тому, для того, щоб слова, що складаються лише з великих літер або завжди починаються з великої літери, саме в такому вигляді і потрапили до набору слів, для яких виконується підрахунок частот, і, відповідно, саме в такому вигляді були записані до файлу, що містить частоти N-грам слів, програмою для

кожного з таких слів виконується перевірка, чи зустрічаються у тексті варіанти написання цього ж самого слова, але починаючи з малої літери. Якщо такі варіанти написання відсутні, або ж кількість таких варіантів написання суттєво менша за той, що розглядається, то слово записується до файлу, що містить частоти *N*-грам слів, саме у такому вигляді – з першою великою літерою або з усіма великими літерами тощо.

3.3.2. Модуль зчитування частот N-грам слів з файлів та запису їх у файли

Модуль зчитування частот *N*-грам слів з файлів та запису їх у файли призначений для збереження у файлах частот *N*-грам, отриманих як в результаті попереднього оброблення деякого текстового корпусу, так і безпосередньо в процесі здійснення введення тексту користувачем, з метою пристосування до особливостей введення тексту цим користувачем, а також для зчитування зі створених раніше файлів даних про частоти *N*-грам слів, отриманих в обох зазначених вище випадках. Необхідність запису та зчитування цих даних з файлу полягає у тому, що таким чином забезпечується збереження цих даних після повторного запуску роботи програмних засобів.

Даний модуль містить всього два методи – метод save_n_grams_frequencies для запису даних про частоти *N*-грам слів у файли та метод load_n_grams_frequencies для зчитування цих даних. Дані про частоти уніграм, біграм та триграм слів зберігаються відповідно у файлах з назвами n_grams_1.txt, n_grams_2.txt та n_grams_3.txt, причому дані, отримані в результаті оброблення текстового корпусу, розміщуються у директорії з назвою n_grams, а дані, отримані в процесі введення тексту користувачем, розміщуються у директорії з назвою n_grams_user. Таке розрізнення даних про частоти *N*-грам слів, отриманих з двох різних джерел дозволяє в подальшому певним чином надавати перевагу саме даним, отриманим в процесі введення тексту користувачем.

Дані про частоти *N*-грам слів у файлах зберігаються у текстовому форматі. Дані про частоти різних *N*-грам розміщені у окремих рядках для забезпечення можливості здійснення більш зручного перегляду вмісту цих файлів під час розроблення та тестування даних програмних засобів. Дані про частоту кожної з *N*-грам, в свою чергу, записуються у форматі «слово:частота» для уніграм, «слово,слово:частота» для біграм, та «слово,слово,слово:частота» для триграм. Такий формат дозволяє як зручний запис даних про частоти *N*-грам слів, так і їх зчитування. Крім того, такий формат є достатньо зручним для перегляду під час розроблення та тестування програмних засобів. При здійсненні запису у файли частот *N*-грам слів, отриманих в результаті оброблення текстового корпусу, що складається з 86390 слів, було отримано три файли, сумарний обсяг яких становив близько 1,35 МБ, що в умовах дуже широких можливостей ресурсів сучасних мобільних пристроїв можна вважати прийнятним обсягом пам’яті для збереження даних про частоти *N*-грам слів.

3.3.3. «Обгортка» над модулем генерування списку пропозицій для введення

Модуль, що виступає у ролі обгортки над модулем генерування списку пропозицій для введення, призначений для виконання попереднього оброблення тексту, введеного користувачем, перед тим як програма здійснюватиме безпосередньо генерування списку пропозицій для введення, а також для виклику відповідних методів з модуля генерування списку пропозицій для введення в залежності від тексту, поданого на вхід даного модуля.

Даний модуль містить лише один метод – метод create_propositions, на вхід якого подаються дані про частоти *N*-грам, отримані як в результаті оброблення текстового корпусу, так і в процесі введення тексту користувачем, а також текст, введений користувачем на даний момент.

Попереднє оброблення тексту, введеного користувачем, полягає у розбивці цього тексту на токени, зрозумілі модулю генерування списку

пропозицій для введення, а також у виборі необхідного методу з модуля генерування списку пропозицій для введення для безпосередньо створення цього списку пропозицій.

Оскільки при генерування N-грам слів враховуються тільки уніграми, біграми та триграми слів, тобто максимальним значенням N в даному випадку є 3, до кінцевого набору токенів, що передається на вхід модуля генерування списку пропозицій для введення, додаються тільки три останні токени, якщо їх кількість у отриманому набору токенів є більшою або рівною 3, а у випадку ж, якщо кількість токенів у отриманому набору токенів є меншою за 3, до кінцевого набору токенів, що передається на вхід модуля генерування списку пропозицій для введення, додаються всі токени з отриманого набору токенів.

Процес розбивки тексту на токени відбувається аналогічним чином до процесу розбивки на токени під час генерування N-грам слів у модулі генерування N-грам слів та підрахунку їх частот на основі деякого текстового корпусу. Однак, в даному випадку, окрім безпосередньо розбивки на токени, здійснюється також вибір необхідного методу для створення списку пропозицій. Такий вибір здійснюється в залежності від двох факторів:

1) наявність або відсутність пробілу безпосередньо після всіх слів, що на даний момент містяться у тексті, введеному користувачем;
2) наявність або відсутність у самому кінці тексту, введеного користувачем, слів, що послідовно розміщуються одне за одним і розділені пробілами.

Так, в залежності від комбінації цих факторів можливі такі дії програмних засобів:

1. Генерування списку пропозицій для введення на основі біграм або триграм, введення останнього слова яких користувачем не було розпочато. Така ситуація можлива у тому випадку, коли

користувач послідовно ввів одне або більше слів, розділених між собою пробілами, після чого ввів пробіл і готовий до введення нового слова. Наприклад, поточний текст, введений користувачем, у даному випадку може мати вигляд «My name |», де символ «|» демонструє поточне місцезнаходження курсора. Так, в такому випадку першими двома словами, що входять до складу триграми, є слова «my» та «name». Відповідно, при здійсненні генерування списку пропозицій для введення в такому випадку необхідним є врахування вже введених частин N-грами при пошуку її можливих закінчень та створенні списку пропозицій для введення відповідно до частот N-грам, що мають ці закінчення.

2. Генерування списку пропозицій для введення у випадку, коли введення першого слова нової послідовності слів користувачем не було розпочато. Така ситуація можлива у тому випадку, коли користувач ще не розпочав введення тексту взагалі, або ж закінчив попередньо введену послідовність слів розділовим знаком або числом тощо, тобто символом чи послідовністю символів, що не є словом, після чого ввів пробіл і готовий до введення нового слова. Наприклад, поточний текст, введений користувачем, у даному випадку може мати вигляд «That was a good day. |», де символ «|» демонструє поточне місцезнаходження курсора. Так, в такому випадку попередню послідовність слів було закінчено крапкою, і, відповідно, користувач готовий до введення нової послідовності. Відповідно, при здійсненні генерування списку пропозицій для введення в такому випадку попередня закінчена послідовність слів не враховується, і генерування списку пропозицій для введення здійснюється лише на основі частот усіх наявних унірам слів.

3. Генерування списку пропозицій для введення на основі уніграм, біграм або триграм, введення останнього слова яких користувачем вже було розпочато. Така ситуація можлива у тому випадку, коли користувач послідовно ввів одне або більше слів, розділених між собою пробілами, або не вводив їх, після чого розпочав введення нового слова. Наприклад, поточний текст, введений користувачем, у даному випадку може мати вигляд «How are yo|», де символ «|» демонструє поточне місцезнаходження курсора. Так, в такому випадку першими двома словами, що входять до складу триграми, є слова «how» та «are». Введення ж третього слова ще не було закінчено, тому в даному випадку, необхідним є як врахування попередньо введених слів, так і врахування вже введеного користувачем початку нового слова.
4. Генерування порожнього списку пропозицій для введення. Така ситуація можлива у тому випадку, коли користувач розпочав введення розділових знаків або чисел тощо, тобто послідовності символів, що не є словом, після чого ще не ввів пробіл, і, таким чином, не закінчив введення послідовності символів, що не є словом. Відповідно, в такому випадку генерування списку пропозицій для введення не відбувається.

3.3.4. Модуль генерування списку пропозицій для введення

Модуль генерування списку пропозицій для введення на основі тексту, який вводить користувач, а також даних про частоти *N*-грам слів, є одним з основних модулів, що входять до складу розроблених програмних засобів. Він містить безпосередньо реалізацію методів, що здійснюють генерування списку пропозицій для введення на основі даних про частоти *N*-грам слів, отриманих як в результаті оброблення текстового корпусу, так і в процесі введення тексту користувачем, а також тексту, введеного

користувачем, який було попередньо оброблено у модулі, що виступає у ролі обгортки над даним модулем.

Як було зазначено в описі модуля, що виступає у ролі обгортки над даним модулем, генерування списку пропозицій для введення може відбуватися за допомогою чотирьох різних методів, вибір яких відбувається в залежності від наявності або відсутності пробілу безпосередньо після всіх слів, що на даний момент містяться у тексті, введеному користувачем, а також від наявності або відсутності у самому кінці тексту, введеного користувачем, слів, що послідовно розміщуються одне за одним і розділені пробілами.

Так, у випадку, коли користувач послідовно ввів одне або більше слів, розділених між собою пробілами, а після цього ввів пробіл, модулем, що виступає у ролі обгортки над даним модулем, відбувається виклик методу create_propositions_new_n_gram – методу генерування списку пропозицій для введення на основі біграм або триграм, введення останнього слова яких користувачем не було розпочато.

У випадку, коли користувач ще не розпочав введення тексту взагалі, або ж закінчив попередньо введену послідовність слів розділовим знаком або числом тощо, тобто символом чи послідовністю символів, що не є словом, а після цього ввів пробіл, модулем, що виступає у ролі обгортки над даним модулем, відбувається виклик методу create_propositions_new_unigram – методу генерування списку пропозицій для введення у випадку, коли введення першого слова нової послідовності слів користувачем не було розпочато.

В свою чергу, у випадку, коли користувач послідовно ввів одне або більше слів, розділених між собою пробілами, або не вводив їх, після чого розпочав введення нового слова, відбувається виклик методу create_propositions_n_gram – методу генерування списку пропозицій для введення на основі унігram, біграм або триграм, введення останнього слова яких користувачем вже було розпочато.

Нарешті, у випадку, коли користувач розпочав введення розділових знаків або чисел тощо, тобто послідовності символів, що не є словом, після чого ще не ввів пробіл, і, таким чином, не закінчив введення послідовності символів, що не є словом, відбувається виклик методу create_propositions_nothing – методу генерування порожнього списку пропозицій для введення.

Реалізація методу генерування списку пропозицій для введення на основі біграм або триграм, введення останнього слова яких користувачем не було розпочато – методу create_propositions_new_n_gram передбачає такі дії:

1. Список токенів, поданих на вхід методу, оброблюється таким чином, щоб врахувати регістр кожного зі слів. Так, у тому випадку, якщо слово, що міститься у списку токенів, відсутнє у поточному написанні серед слів, дані щодо частот уніграм яких наявні у програмі, програмою відбувається пошук серед цих слів варіанту написання слова у нижньому регістрі, варіанту написання слова, починаючи з великої літери, а також варіанту написання слова у верхньому регістрі. В разі, якщо такого слова навіть у різних варіантах написання все одно не було знайдено серед слів, дані щодо частот уніграм яких наявні у програмі, слово додається до обробленого списку токенів у поточному написанні.
2. Серед N-грам слів, отриманих в процесі введення тексту користувачем, відбувається пошук N-грам слів таких, першими N–1 елементами яких є слова, що містяться у обробленому списку токенів, після чого отриманий набір N-грам слів сортується відповідно до частот цих N-грам слів і його відсортовані елементи додаються до списку пропозицій для введення.

3. У випадку, якщо поточний список пропозицій для введення містить менше елементів, ніж встановлена максимальна кількість елементів списку пропозицій для введення, поточний список пропозицій для введення доповнюється шляхом здійснення аналогічного пошуку N-грам слів серед N-грам слів, отриманих в результаті оброблення текстового корпусу, і додавання знайдених і відсортованих за частотою елементів в кінець поточного списку пропозицій для введення. Слід зазначити, що повторне додавання слів, що вже містяться у списку пропозицій для введення, не відбувається.
4. У випадку, якщо поточний список пропозицій для введення все одно містить менше елементів, ніж встановлена максимальна кількість елементів списку пропозицій для введення, а оброблений список токенів складається більше, ніж з одного елемента, з обробленого списку токенів вилучається перший елемент, і відбувається повторний пошук, але N в даному випадку зменшується на одиницю. У випадку ж, якщо поточний список пропозицій для введення все одно містить менше елементів, ніж встановлена максимальна кількість елементів списку пропозицій для введення, а оброблений список токенів складається з одного елемента, або ж поточний список пропозицій для введення складається з такої кількості елементів, що рівна або перевищує максимальну кількість елементів списку пропозицій для введення, генерування списку пропозицій для введення вважається завершеним.
5. У випадку, якщо отриманий список пропозицій для введення складається з такої кількості елементів, що перевищує максимальну кількість елементів списку пропозицій для введення, до кінцевого списку пропозицій для введення додаються лише перші елементи отриманого списку таким

чином, щоб кількість елементів кінцевого списку дорівнювала максимальній кількості елементів списку пропозицій для введення.

Реалізація методу генерування списку пропозицій для введення у випадку, коли введення першого слова нової послідовності слів користувачем не було розпочато – методу create_propositions_new_unigram передбачає такі дії:

1. Список усіх уніграм слів, отриманих в процесі введення тексту користувачем, сортується відповідно до частот цих уніграм слів і його відсортовані елементи додаються до списку пропозицій для введення.
2. У випадку, якщо отриманий список пропозицій для введення містить менше елементів, ніж встановлена максимальна кількість елементів списку пропозицій для введення, список усіх уніграм слів, отриманих в результаті оброблення текстового корпусу, сортується відповідно до частот цих уніграм слів і його відсортовані елементи додаються до списку пропозицій для введення.
3. З отриманого списку пропозицій для введення вилучаються дублікати.
4. У випадку, якщо отриманий відфільтрований список пропозицій для введення складається з такої кількості елементів, що перевищує максимальну кількість елементів списку пропозицій для введення, до кінцевого списку пропозицій для введення додаються лише перші елементи отриманого списку таким чином, щоб кількість елементів кінцевого списку дорівнювала максимальній кількості елементів списку пропозицій для введення.

Реалізація методу генерування списку пропозицій для введення на основі уніграм, біграм або триграм, введення останнього слова яких

користувачем вже було розпочато – методу create_propositions_n_gram передбачає такі дії:

1. Отримання списку можливих початків слів в залежності від розподілу літер між блоками з відповідного модуля.
2. Список токенів, поданих на вхід методу, оброблюється таким чином, щоб врахувати регістр кожного зі слів. Дані дії здійснюються аналогічно до дій, що здійснюються при виконанні методу генерування списку пропозицій для введення на основі біграм або триграм, введення останнього слова яких користувачем не було розпочато.
3. З отриманого списку можливих початків слів отримується список слів, кожне з яких відповідає одному з отриманих можливих початків слів, і дані щодо частот уніграм яких були отримані в процесі введення тексту користувачем.
4. У випадку, якщо список оброблених токенів містить лише один елемент, отриманий список слів сортується відповідно до частот уніграм цих слів і відсортовані елементи цього списку додаються до списку пропозицій для введення із врахуванням максимальної кількості елементів списку пропозицій для введення, аналогічно до дій, що здійснюються при виконанні методу генерування списку пропозицій для введення у випадку, коли введення першого слова нової послідовності слів користувачем не було розпочато.
5. У випадку ж, якщо список оброблених токенів містить більше одного елементу, для кожного слова з отриманого списку слів отримуються дані про частоти *N*-грам слів, останнім елементом яких є це слово, а першими елементами яких є відповідно елементи списку оброблених токенів за винятком останнього елемента. Отриманий список слів сортується відповідно до отриманих даних про частоти *N*-грам слів, і його

відсортовані елементи додаються до списку пропозицій для введення, аналогічно до дій, що здійснюються при виконанні методу генерування списку пропозицій для введення на основі біграм або триграм, введення останнього слова яких користувачем не було розпочато.

6. У випадку, якщо отриманий список пропозицій для введення містить менше елементів, ніж встановлена максимальна кількість елементів списку пропозицій для введення, аналогічним чином отримується список пропозицій для введення, отриманий з використанням даних про частоти N-грам слів, отриманих в результаті оброблення текстового корпусу, і елементи цього списку додаються в кінець списку, отриманого з використанням даних про частоти N-грам слів, отриманих в процесі введення тексту користувачем.
7. З отриманого списку пропозицій для введення вилучаються дублікати.
8. У випадку, якщо отриманий відфільтрований список пропозицій для введення складається з такої кількості елементів, що перевищує максимальну кількість елементів списку пропозицій для введення, до кінцевого списку пропозицій для введення додаються лише перші елементи отриманого списку таким чином, щоб кількість елементів кінцевого списку дорівнювала максимальній кількості елементів списку пропозицій для введення.

Слід також зазначити, що в методі, описаному вище, перший елемент списку можливих початків слів, має перевагу перед усіма іншими елементами цього списку. Така перевага полягає у тому, що оброблення списку слів, кожне з яких відповідає першому з отриманих можливих початків слів, здійснюється окремо від оброблення списку слів, кожне з яких відповідає одному з інших можливих початків слів, окрім першого.

Так, слова, що відповідають першому з отриманих можливих початків слів, розташовуватимуться першими у кінцевому списку пропозицій для введення навіть у тому випадку, якщо частоти N-грам інших слів будуть вищими.

Реалізація методу генерування порожнього списку пропозицій для введення – методу create_propositions_nothing передбачає лише створення порожнього списку пропозицій для введення.

3.3.5. Модуль генерування можливих початків слів в залежності від розподілу літер між блоками

Модуль генерування можливих початків слів в залежності від розподілу літер між блоками також є одним з основних модулів, що входять до складу розроблених програмних засобів. Крім того, саме у цьому модулі реалізована основна ідея модифікованого методу предиктивного введення тексту на основі N-грам – застосування блоків літер, згрупованих відповідно до розташування цих літер на клавіатурі.

При запуску програми у даному модулі здійснюється зчитування даних про розподіл літер між блоками з допоміжного текстового файлу letter_blocks.txt. Цей файл містить набір рядків, кожен з яких відповідає певному блоку. Першою літерою у рядку є літера, що є центром блоку, іншими літерами є літери, що разом із першою літерою входять до цього блоку. Так, приклад вмісту цього файлу для американського варіанту QWERTY-клавіатури наведено на рис. 15.

Також, даний модуль містить метод create_candidates – метод генерування можливих початків слів, що приймає на вхід введену користувачем послідовність символів – токен, а також дані про частоти N-грам слів, отримані як в процесі введення тексту користувачем, так і в результаті оброблення текстового корпусу.

При виклику даного методу здійснюється створення списку слів, дані щодо частот уніграм яких відомі програмі. Після цього створюється

початковий список можливих послідовностей, а зі списку слів вилучаються ті, що не відповідають.

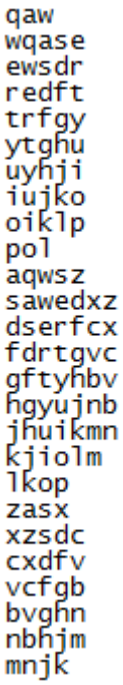


Рис. 15. Приклад вмісту текстового файлу, у якому міститься інформація про розподіл літер між блоками, для американського варіанту QWERTY-клавіатури

Після цього для кожної літери з отриманого на вхід методу токену здійснюється отримання списку літер, що входять до блоку, центром якої є дана літера. Для кожної з літер блоку здійснюється пошук слів, які починаються з даної літери, серед слів, дані щодо частот уніграм яких відомі програмі. Слова, що не увійшли до результатів даного пошуку, вилучаються зі списку можливих слів, а літери, з яких не починається жодне слово, відповідно, вилучаються зі списку можливих слів.

Після цього для першої літери з отриманого на вхід методу токену здійснюється отримання списку літер, що входять до блоку, центром якої є дана літера. Для кожної з літер блоку здійснюється пошук слів, які починаються з даної літери, серед слів, дані щодо частот уніграм яких відомі програмі. Слова, які починаються з однієї з літер блоку, додаються

до початкового списку можливих слів, а літери блоку, з яких починається хоча б одне слово, додаються до початкового списку можливих послідовностей таким чином, що кожна літера є початком окремої послідовності.

Далі, для кожної наступної літери з отриманого на вхід методу токену так само здійснюється отримання списку літер, що входять до блоку, центром якої є дана літера. Для кожної послідовності зі списку можливих послідовностей здійснюється пошук слів, які починаються з даної послідовності, із додаванням до цієї послідовності почергово кожної з літер блоку. Слова, які починаються з даної послідовності із додаванням деякої літери блоку, додаються до нового списку можливих слів, виключаючи таким чином із попереднього списку можливих слів слова, що не відповідають жодній з комбінацій послідовності та літери блоку. Комбінації ж послідовності та літери блоку, з яких починається хоча б одне слово, додаються до нового списку можливих послідовностей.

Таким чином, після закінчення оброблення останньої літери отриманого на вхід методу токену отримується список можливих початків слів, з кожного з яких починається хоча б одне слово зі списку слів, дані щодо частот уніграм яких відомі програмі, з урахуванням можливих натискань користувачем літер, що входять до того самого блоку, що й бажана літера.

3.3.6. Модуль, що здійснює оновлення частот N-грам слів на основі тексту, який вводить користувач

Модуль, що здійснює оновлення частот *N*-грам слів на основі тексту, який вводить користувач, призначений для врахування специфіки введення тексту тим чи іншим користувачем під час наступних створень списків пропозицій для введення.

Даний метод містить лише один метод – метод updated_user_frequencies, що приймає на вхід поточні дані про частоти *N*-грам слів, отримані як в процесі введення тексту користувачем, так і в

результаті оброблення текстового корпусу, а також останній введений користувачем рядок тексту.

Після цього відбувається розбиття текстового рядку на токени аналогічно тому, як це здійснюється у модулі генерування N-грам слів та підрахунку їх частот на основі деякого текстового корпусу. Далі з отриманого набору токенів так само виокремлюється список слів, на основі якого генерується набір уніграм та відбувається підрахунок їх частот. Аналогічно, на основі отриманого набору токенів генерується набір біграм і триграм та відбувається підрахунок їх частот.

Слід зазначити, що, так само як і в модулі генерування N-грам слів та підрахунку їх частот на основі деякого текстового корпусу, в даному модулі при здійсненні генерування уніграм, біграм та триграм слів здійснюється врахування спеціальним чином регістру слів, що підлягають обробленню.

Результатом роботи методу є оновлені дані про частоти N-грам слів, отримані в процесі введення тексту користувачем.

3.3.7. Модуль інтерфейсу взаємодії із користувачем

Модуль інтерфейсу взаємодії із користувачем надає можливість користувачеві здійснювати введення тексту із відображенням йому списку пропозицій для введення, які користувач може як застосовувати, так і не застосовувати, продовжуючи самостійне введення тексту.

Інтерфейс користувача складається з блоку виведення пропозицій для введення та блоку для введення тексту. Вибір бажаної пропозиції для введення здійснюється за допомогою натискання відповідно клавіш F1, F2 та F3. У блоці для введення тексту відображається не більше п’яти останніх рядків введеного тексту. При натисканні клавіші Enter відбувається перехід на новий рядок та оновлення даних про частоти N-грам слів на основі попереднього рядка. При натисканні клавіші Escape відбувається вихід з програми. Приклад відображення інтерфейсу користувача наведено на рис. 16.

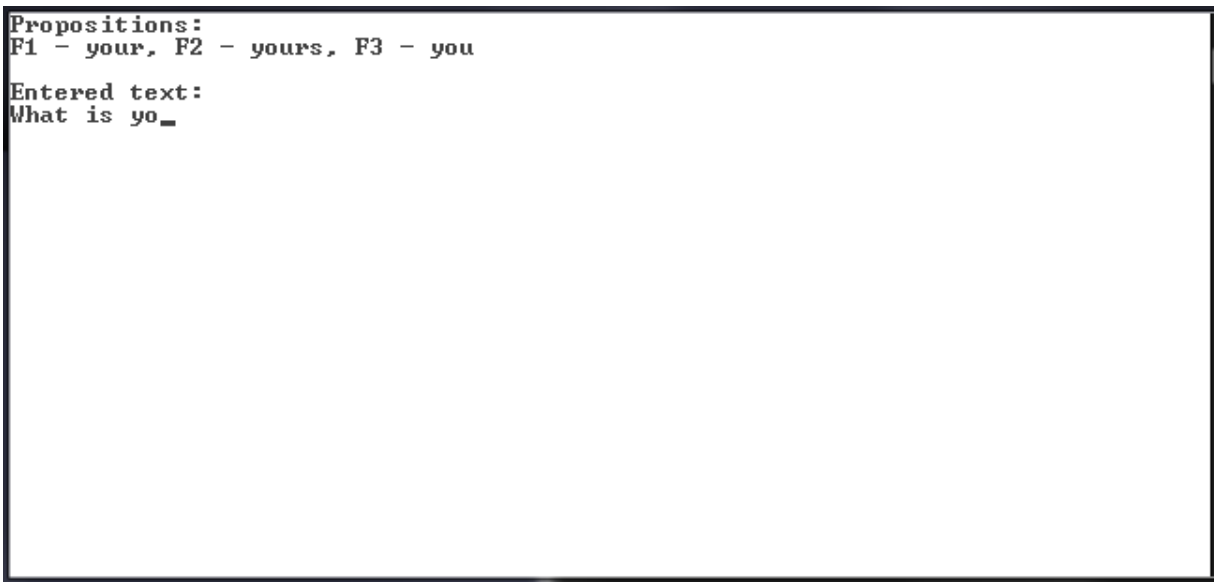


Рис. 16. Приклад відображення інтерфейсу користувача розроблених програмних засобів

При натисканні інших клавіш відбувається їх введення у блоці для введення тексту, а також створення і відображення нового списку пропозицій для введення.

При виборі деякої пропозиції зі списку поточне слово, введене користувачем, замінюється на обране, після чого у блоці для введення тексту в кінець автоматично додається пробіл і відбувається створення і відображення нового списку пропозицій для введення.

3.4. Опис структур даних

Для зберігання даних про частоти *N*-грам слів у пам’яті програми використовується словник (dict) – один із типів даних, вбудованих в мові програмування Python.

Для зберігання даних частот *N*-грам слів, отриманих в процесі введення тексту користувачем, та частот *N*-грам слів, отриманих в результаті оброблення текстового корпусу, використовуються два окремих словника, що мають однакову структуру.

Ключами таких словників є цілі числа, що відповідають числу N для тієї чи іншої *N*-грами – відповідно 1, 2 або 3. Значеннями таких словників,

в свою чергу, є також словники, ключами яких є власне *N*-грами, а значеннями – їх частоти.

Уніграми представлені у таких словниках у вигляді рядка, а біграми та триграми – у вигляді незмінних списків (tuple). Частоти уніграм, в свою чергу, представлені у вигляді цілих чисел. І рядок, і незмінний список, і ціле число, також як і словник, є типами даних, вбудованими в мові програмування Python.

Приклад словника, що містить дані про частоти *N*-грам слів, наведено на рис. 17.

```
{
        1: {
                'how': 6,
                'you': 12,
                'are': 15
        },
        2: {
                ('how', 'are'): 3,
                ('are', 'you'): 5
        },
        3: {
                ('how', 'are', 'you'): 2
        }
}
```

Рис. 17. Приклад словника, що містить дані про частоти *N*-грам слів

Дані про розподіл літер між блоками також зберігаються у пам’яті програми за допомогою словника. Ключами такого словника є рядки, що містять літеру, яка є центром блоку, а значеннями – незмінні списки, які містять літери, що також входять до цього блоку. Кожна літера цього списку також представлена у вигляді рядка.

Приклад словника, що містить дані про розподіл літер між блоками для американського варіанту QWERTY-клавіатури, наведено на рис. 18.

Слід зазначити, що використання словників дозволяє досягати прийнятних часових показників роботи програмних засобів, оскільки така структура даних дозволяє здійснювати операції читання та запису даних достатньо швидко. Крім того, вона є інтуїтивно зрозумілою і достатньо

легкою у використанні, що, відповідно, спрощує процес розроблення програмних засобів.

```
{
        'a': ('a', 'q', 'w', 's', 'z'),
        'c': ('c', 'x', 'd', 'f', 'v'),
        'b': ('b', 'v', 'g', 'h', 'n'),
        'e': ('e', 'w', 's', 'd', 'r'),
        'd': ('d', 's', 'e', 'r', 'f', 'c', 'x'),
        'g': ('g', 'f', 't', 'y', 'h', 'b', 'v'),
        'f': ('f', 'd', 'r', 't', 'g', 'v', 'c'),
        'i': ('i', 'u', 'j', 'k', 'o'),
        'h': ('h', 'g', 'y', 'u', 'j', 'n', 'b'),
        'k': ('k', 'j', 'i', 'o', 'l', 'm'),
        'j': ('j', 'h', 'u', 'i', 'k', 'm', 'n'),
        'm': ('m', 'n', 'j', 'k'),
        'l': ('l', 'k', 'o', 'p'),
        'o': ('o', 'i', 'k', 'l', 'p'),
        'n': ('n', 'b', 'h', 'j', 'm'),
        'q': ('q', 'a', 'w'),
        'p': ('p', 'o', 'l'),
        's': ('s', 'a', 'w', 'e', 'd', 'x', 'z'),
        'r': ('r', 'e', 'd', 'f', 't'),
        'u': ('u', 'y', 'h', 'j', 'i'),
        't': ('t', 'r', 'f', 'g', 'y'),
        'w': ('w', 'q', 'a', 's', 'e'),
        'v': ('v', 'c', 'f', 'g', 'b'),
        'y': ('y', 't', 'g', 'h', 'u'),
        'x': ('x', 'z', 's', 'd', 'c'),
        'z': ('z', 'a', 's', 'x')
}
```

Рис. 18. Приклад словника, що містить дані про розподіл літер між блоками для американського варіанту QWERTY-клавіатури

4. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1. Характеристика тестових наборів даних

4.1.1. Характеристика текстових корпусів

Для тестування запропонованої модифікації методу предиктивного введення тексту на основі *N*-грам за допомогою розробленого програмного забезпечення було використано дані з п’яти текстових корпусів англійської мови, на основі яких відбувалось генерування початкового списку *N*-грам та здійснювався підрахунок їх частот. Опис використаних текстових корпусів наведено у табл. 3.

Таблиця 3

Опис текстових корпусів, використаних для тестування запропонованої модифікації методу предиктивного введення тексту на основі *N*-грам

№	Назва	Опис
1	The Corpus of Contemporary American English (COCA) [29]	Корпус американської англійської мови, один з найбільш часто використовуваних текстових корпусів англійської мови. Складається з текстів 1990-2017 років та містить розмовні тексти, художню літературу, тексти газет та популярних журналів, а також наукові тексти.
2	The Corpus of Historical American English (COHA) [29]	Корпус історичної англійської мови, що складається з текстів 1810-2000 років. Містить художню літературу, науково-популярні тексти, тексти популярних журналів та тексти газет.
3	Corpus of Global Web-Based English [29]	Корпус, що складається з текстів веб-сторінок, здебільшого блогів. Даний корпус містить тексти різних варіантів англійської мови, що використовуються у США, Великій Британії, Австралії, Індії та ще 16-ти країнах.

Продовження табл. 3

№	Назва	Опис
4	Corpus of News on the Web (NOW) [29]	Корпус, що складається з текстів онлайн-журналів та газет 20-ти різних англомовних країн, починаючи з 2010 року і до сьогодні, і постійно оновлюється.
5	The Wikipedia Corpus [29]	Корпус, що містить набір сторінок з популярної інтернет-енциклопедії Вікіпедія, зроблений у 2014 році.

Слід зазначити, що в процесі тестування запропонованої модифікації методу предиктивного введення тексту на основі *N*-грам було використано вибірки (англ. «sample») з зазначених вище текстових корпусів, кожна з яких була отримана з відповідного текстового корпусу випадковим чином. Кожна з таких вибірок містить від 1,7 млн (текстові корпуси №1 та №4) до 3,6 млн слів (текстовий корпус №2).

4.1.2. Характеристика текстів для введення

Для тестування запропонованої модифікації методу предиктивного введення тексту на основі *N*-грам за допомогою розробленого програмного забезпечення було використано два тексти, введення яких відбувалося за допомогою інтерфейсу користувача. Кожен з цих двох текстів містить близько 300-400 символів.

Перший текст містить переважно загальну лексику, що застосовується при написанні статей, публікацій тощо, а другий – переважно розмовну лексику, що застосовується, в свою чергу, у повсякденному спілкуванні. Зміст даних текстів наведено у табл. 4.

Також, для кожного з текстів було створено додатково п’ять текстів, у яких деякі літери у словах було замінено на сусідні літери на клавіатурі, що відображає помилкові натискання сусідніх літер, з метою тестування методу в умовах таких можливих помилкових натискань. Кожен з цих

п’яти текстів містить відповідно 1%, 5%, 10%, 20% та 25% літер, що були замінені.

Слід зазначити, що з метою забезпечення для всіх текстів однакових умов тестування на початку введення кожного з текстів здійснювалося очищення даних про частоти *N*-грам слів, що були отримані в результаті введення попереднього тексту.

Таблиця 4

Зміст текстів для введення, використаних для тестування запропонованої модифікації методу предиктивного введення тексту на основі *N*-грам

№	Лексика	Зміст
1	Загальна	The Moon is an astronomical body that orbits planet Earth and is Earth's only permanent natural satellite. It is the fifth-largest natural satellite in the Solar System, and the largest among planetary satellites relative to the size of the planet that it orbits (its primary). The Moon is after Jupiter's satellite Io the second-densest satellite in the Solar System among those whose densities are known.
2	Розмовна	How are you, thanks for the message for my birthday! You are welcome. I want to see you as soon as possible for a drink. Perhaps on Saturday in the evening, we could go to the pictures! Ok great but I've got to be right back before midnight. Because I see my grandmother for lunch on Sunday It's ok see you on Saturday 8 PM at home.

4.2. Порівняння отриманих результатів

Для дослідження ефективності запропонованої модифікації методу предиктивного введення тексту на основі *N*-грам у порівнянні з оригінальним методом предиктивного введення тексту на основі *N*-грам

було здійснено тестування розробленого програмного забезпечення з використанням зазначених вище текстових корпусів.

Тестування програмного забезпечення з використанням модифікованого методу здійснювалося із застосуванням даних про розподіл літер між блоками, а оригінального методу, відповідно, – без застосування цих даних.

Оцінка ефективності обох методів здійснювалася за допомогою використання критерію ефективності KSPC. Тестування обох методів з використанням кожного з текстових корпусів та текстів для введення здійснювалося шляхом введення одного і того ж самого тексту із застосуванням пропозицій для введення, а також підрахунку значення KSPC для кожного тестового випадку.

4.2.1. Результати для текстового корпусу №1

Результати, отримані в процесі проведеного тестування з використанням текстового корпусу №1, наведено у табл. 5.

Таблиця 5

Результати, отримані в процесі проведеного тестування з використанням текстового корпусу №1

Відсоток замінених літер	Лексика тексту для введення	Значення KSPC для тексту	
		Оригінальний метод	Модифікований метод
0%	Загальна	0,601	0,616
	Розмовна	0,548	0,575
1%	Загальна	0,645	0,621
	Розмовна	0,584	0,575
5%	Загальна	0,847	0,707
	Розмовна	0,675	0,590

Продовження табл. 5

Відсоток замінених літер	Лексика тексту для введення	Значення KSPC для тексту	
		Оригінальний метод	Модифікований метод
10%	Загальна	0,995	0,783
	Розмовна	0,705	0,596
15%	Загальна	1,094	0,828
	Розмовна	0,783	0,642
20%	Загальна	1,246	0,887
	Розмовна	0,867	0,645

На рис. 19 наведено діаграму порівняння результатів роботи оригінального та модифікованого методу, отриманих з використанням текстового корпусу №1 для тексту із загальною лексикою, а на рис. 20 – із розмовною.

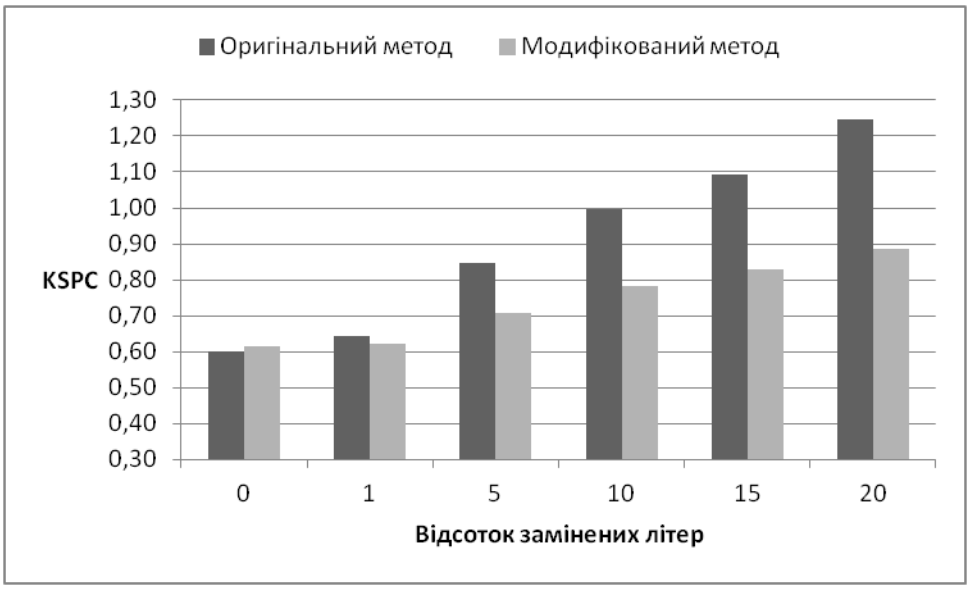


Рис. 19. Порівняння результатів роботи для текстового корпусу №1 і тексту із загальною лексикою

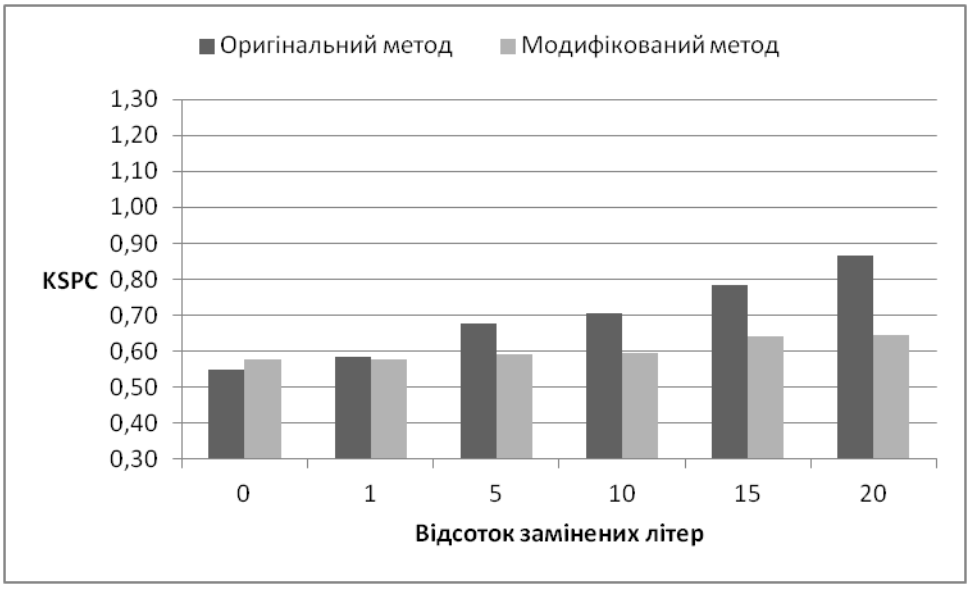


Рис. 20. Порівняння результатів роботи для текстового корпусу №1 і тексту із розмовною лексикою

4.2.2. Результати для текстового корпусу №2

Результати, отримані в процесі проведеного тестування з використанням текстового корпусу №2, наведено у табл. 6.

Таблиця 6

Результати, отримані в процесі проведеного тестування з використанням текстового корпусу №2

Відсоток замінених літер	Лексика тексту для введення	Значення KSPC для тексту	
		Оригінальний метод	Модифікований метод
0%	Загальна	0,571	0,584
	Розмовна	0,587	0,608
1%	Загальна	0,596	0,594
	Розмовна	0,623	0,608
5%	Загальна	0,798	0,702
	Розмовна	0,714	0,620

Продовження табл. 6

Відсоток замінених літер	Лексика тексту для введення	Значення KSPC для тексту	
		Оригінальний метод	Модифікований метод
10%	Загальна	0,946	0,756
	Розмовна	0,762	0,627
15%	Загальна	1,025	0,800
	Розмовна	0,858	0,675
20%	Загальна	1,177	0,810
	Розмовна	0,943	0,681

На рис. 21 наведено діаграму порівняння результатів роботи оригінального та модифікованого методу, отриманих з використанням текстового корпусу №2 для тексту із загальною лексикою, а на рис. 22 – із розмовною.

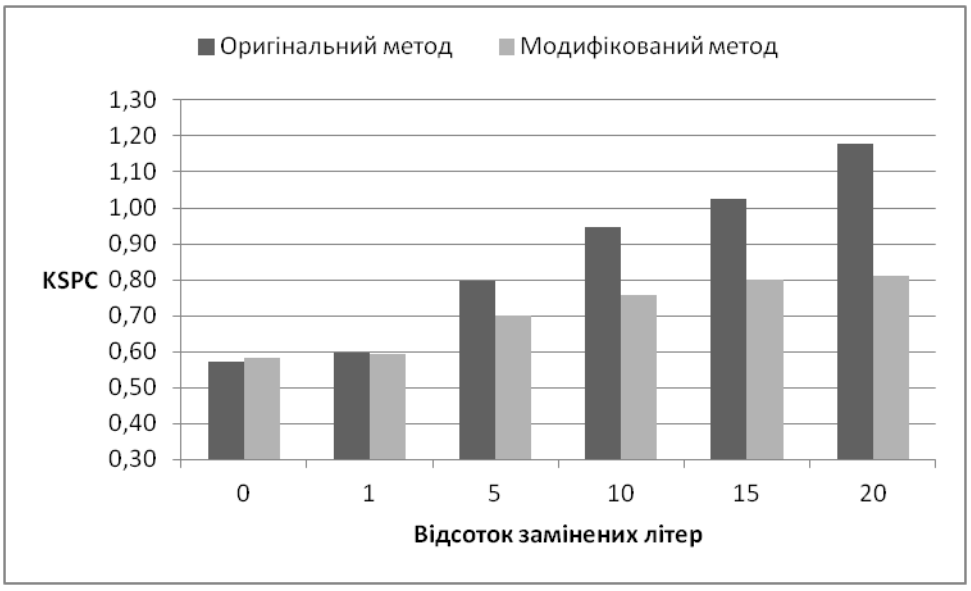


Рис. 21. Порівняння результатів роботи для текстового корпусу №2 і тексту із загальною лексикою

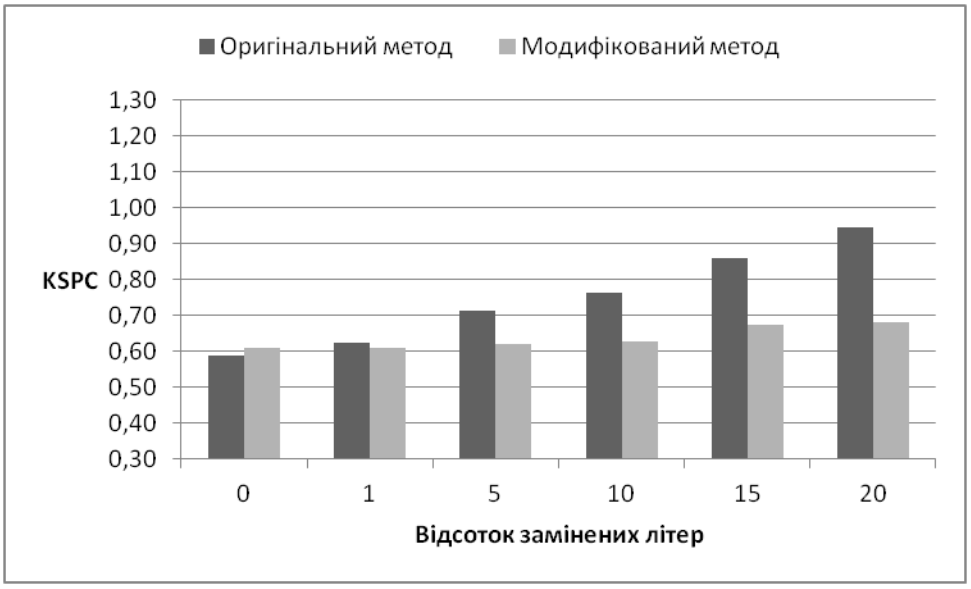


Рис. 22. Порівняння результатів роботи для текстового корпусу №2 і тексту із розмовною лексикою

4.2.3. Результати для текстового корпусу №3

Результати, отримані в процесі проведеного тестування з використанням текстового корпусу №3, наведено у табл. 7.

Таблиця 7

Результати, отримані в процесі проведеного тестування з використанням текстового корпусу №3

Відсоток замінених літер	Лексика тексту для введення	Значення KSPC для тексту	
		Оригінальний метод	Модифікований метод
0%	Загальна	0,586	0,599
	Розмовна	0,587	0,605
1%	Загальна	0,631	0,608
	Розмовна	0,623	0,608
5%	Загальна	0,823	0,712
	Розмовна	0,714	0,630

Продовження табл. 7

Відсоток замінених літер	Лексика тексту для введення	Значення KSPC для тексту	
		Оригінальний метод	Модифікований метод
10%	Загальна	0,970	0,788
	Розмовна	0,762	0,636
15%	Загальна	1,010	0,833
	Розмовна	0,840	0,675
20%	Загальна	1,163	0,892
	Розмовна	0,955	0,684

На рис. 23 наведено діаграму порівняння результатів роботи оригінального та модифікованого методу, отриманих з використанням текстового корпусу №3 для тексту із загальною лексикою, а на рис. 24 – із розмовною.

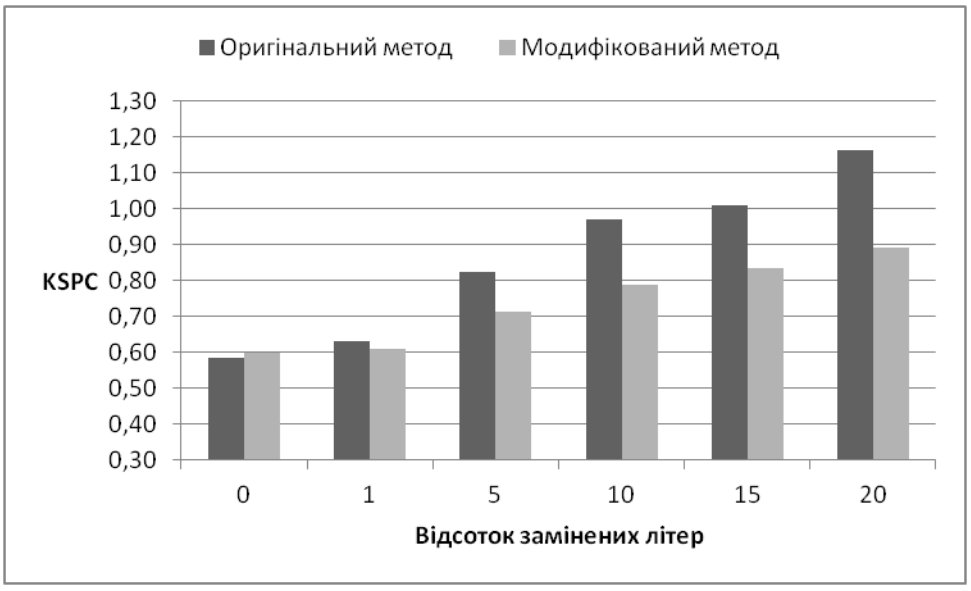


Рис. 23. Порівняння результатів роботи для текстового корпусу №3 і тексту із загальною лексикою

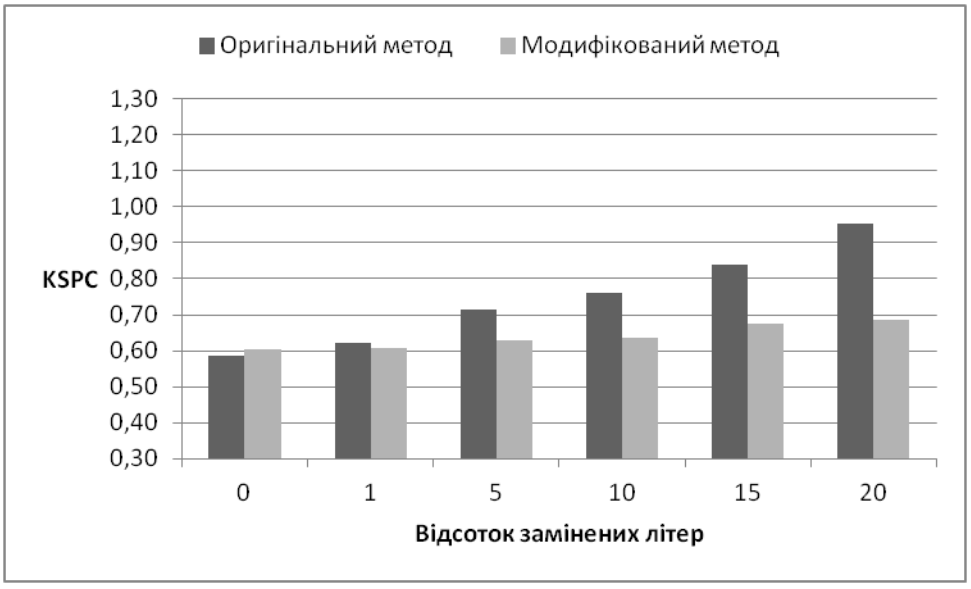


Рис. 24. Порівняння результатів роботи для текстового корпусу №3 і тексту із розмовною лексикою

4.2.4. Результати для текстового корпусу №4

Результати, отримані в процесі проведеного тестування з використанням текстового корпусу №4, наведено у табл. 8.

Таблиця 8

Результати, отримані в процесі проведеного тестування з використанням текстового корпусу №4

Відсоток замінених літер	Лексика тексту для введення	Значення KSPC для тексту	
		Оригінальний метод	Модифікований метод
0%	Загальна	0,589	0,599
	Розмовна	0,560	0,584
1%	Загальна	0,633	0,608
	Розмовна	0,596	0,587
5%	Загальна	0,850	0,727
	Розмовна	0,687	0,605

Продовження табл. 8

Відсоток замінених літер	Лексика тексту для введення	Значення KSPC для тексту	
		Оригінальний метод	Модифікований метод
10%	Загальна	1,007	0,813
	Розмовна	0,723	0,617
15%	Загальна	1,067	0,857
	Розмовна	0,801	0,657
20%	Загальна	1,214	0,911
	Розмовна	0,904	0,666

На рис. 25 наведено діаграму порівняння результатів роботи оригінального та модифікованого методу, отриманих з використанням текстового корпусу №4 для тексту із загальною лексикою, а на рис. 26 – із розмовною.

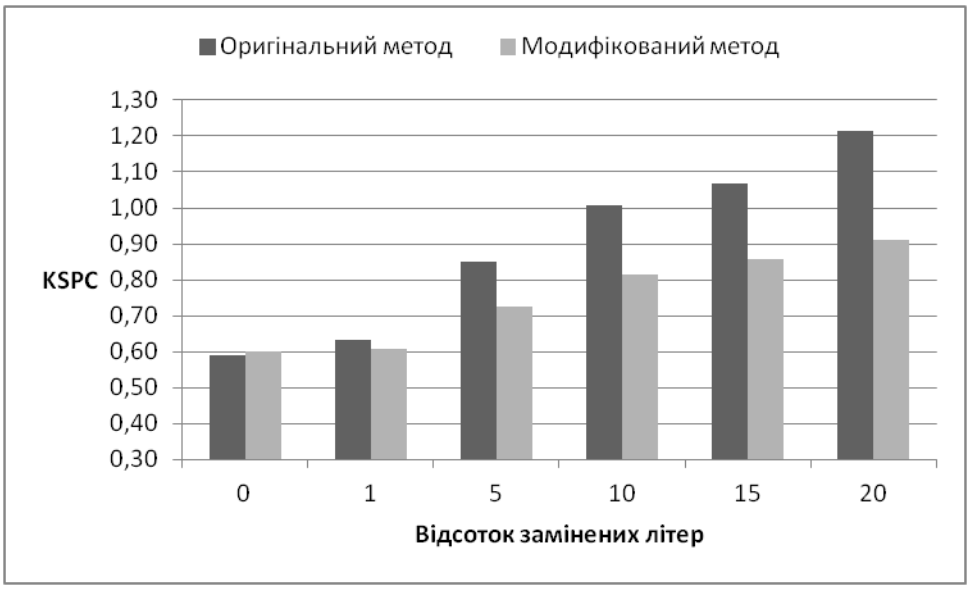


Рис. 25. Порівняння результатів роботи для текстового корпусу №4 і тексту із загальною лексикою

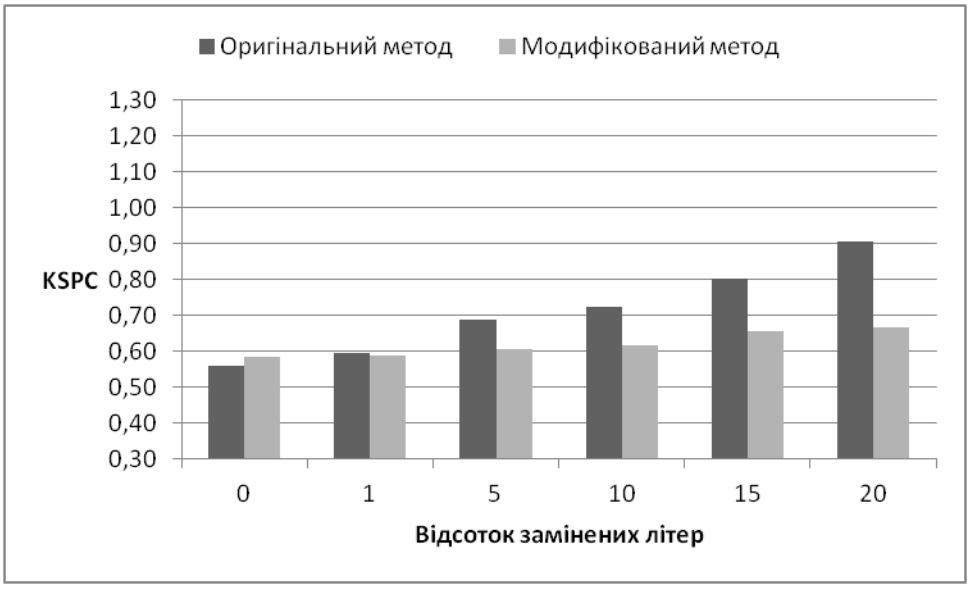


Рис. 26. Порівняння результатів роботи для текстового корпусу №4 і тексту із розмовною лексикою

4.2.5. Результати для текстового корпусу №5

Результати, отримані в процесі проведеного тестування з використанням текстового корпусу №5, наведено у табл. 9.

Таблиця 9

Результати, отримані в процесі проведеного тестування з використанням текстового корпусу №5

Відсоток замінених літер	Лексика тексту для введення	Значення KSPC для тексту	
		Оригінальний метод	Модифікований метод
0%	Загальна	0,557	0,567
	Розмовна	0,663	0,684
1%	Загальна	0,601	0,569
	Розмовна	0,699	0,681
5%	Загальна	0,788	0,638
	Розмовна	0,795	0,696

Продовження табл. 9

Відсоток замінених літер	Лексика тексту для введення	Значення KSPC для тексту	
		Оригінальний метод	Модифікований метод
10%	Загальна	0,946	0,685
	Розмовна	0,861	0,741
15%	Загальна	1,039	0,729
	Розмовна	0,964	0,780
20%	Загальна	1,187	0,734
	Розмовна	1,078	0,858

На рис. 27 наведено діаграму порівняння результатів роботи оригінального та модифікованого методу, отриманих з використанням текстового корпусу №5 для тексту із загальною лексикою, а на рис. 28 – із розмовною.

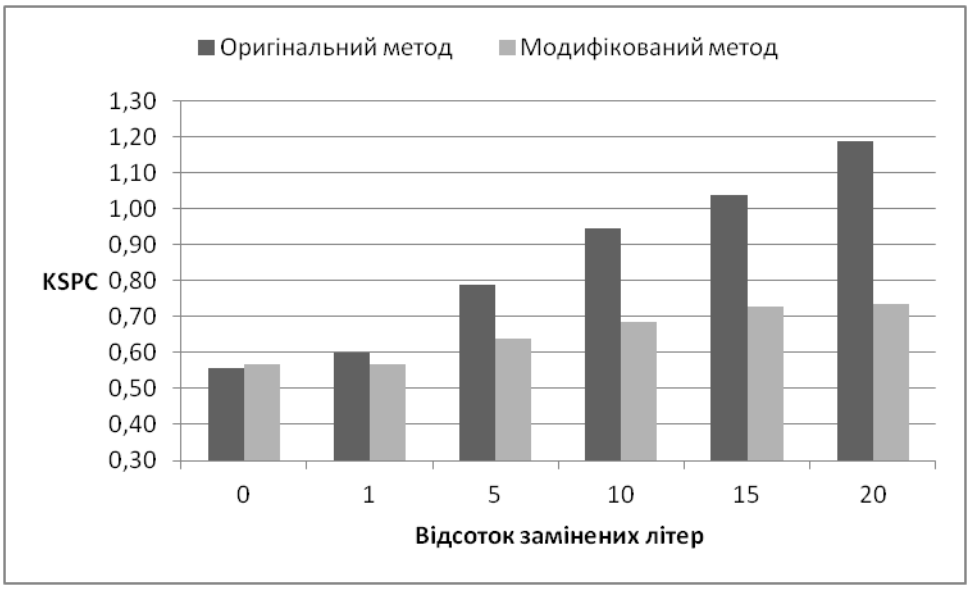


Рис. 27. Порівняння результатів роботи для текстового корпусу №5 і тексту із загальною лексикою

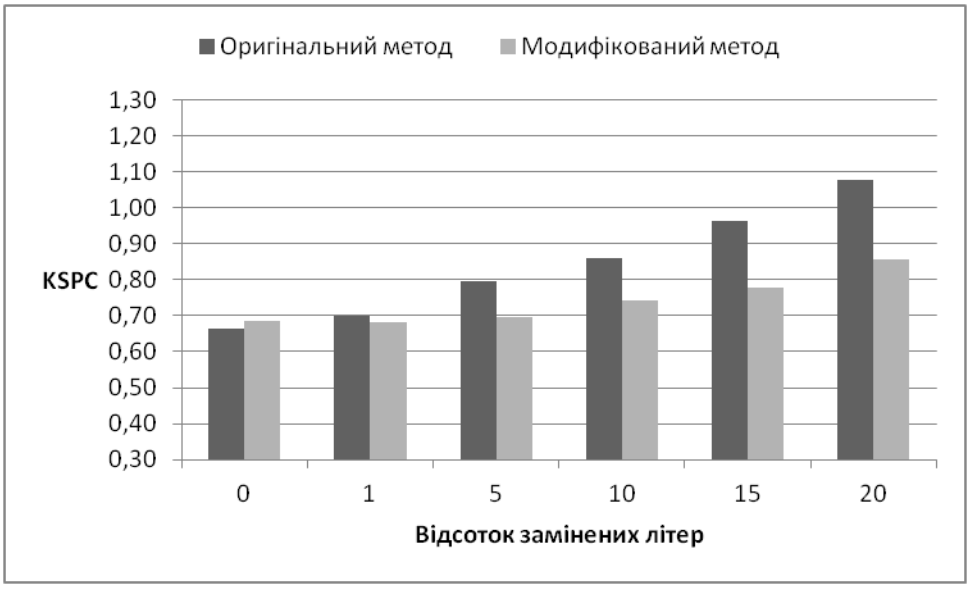


Рис. 28. Порівняння результатів роботи для текстового корпусу №5 і тексту із розмовною лексикою

4.2.6. Аналіз отриманих результатів

Як видно, отримані результати є достатньо схожими для усіх п’ятьох текстових корпусів. При відсутності замінених літер значення KSPC для модифікованого методу для всіх текстових корпусів та з використанням тексту як із загальною лексикою, так і з розмовною, є вищим, ніж для оригінального. Однак, така різниця є незначною і становить в середньому 2,1% для текстів із загальною лексикою та 3,76% – із розмовною.

У всіх інших випадках, а саме, за наявності під час тестування 1%, 5%, 10% 15% та 20% замінених літер, значення KSPC для модифікованого методу для всіх текстових корпусів та з використанням тексту як із загальною лексикою, так і з розмовною, є нижчим, ніж для оригінального. Причому, різниця між значеннями KSPC для цих двох методів збільшується зі збільшенням відсотка замінених літер. Так, у випадку заміни 1% літер значення KSPC для модифікованого методу є нижчим в середньому на 3,41% для текстів із загальною лексикою та на 2,11% – із розмовною, а у випадку заміни 20% літер – на 29,28 та 25,55% відповідно.

На рис. 29 наведено діаграму порівняння результатів роботи оригінального та модифікованого методу в середньому для всіх текстових корпусів і тексту із загальною лексикою, а на рис. 30 – із розмовною.

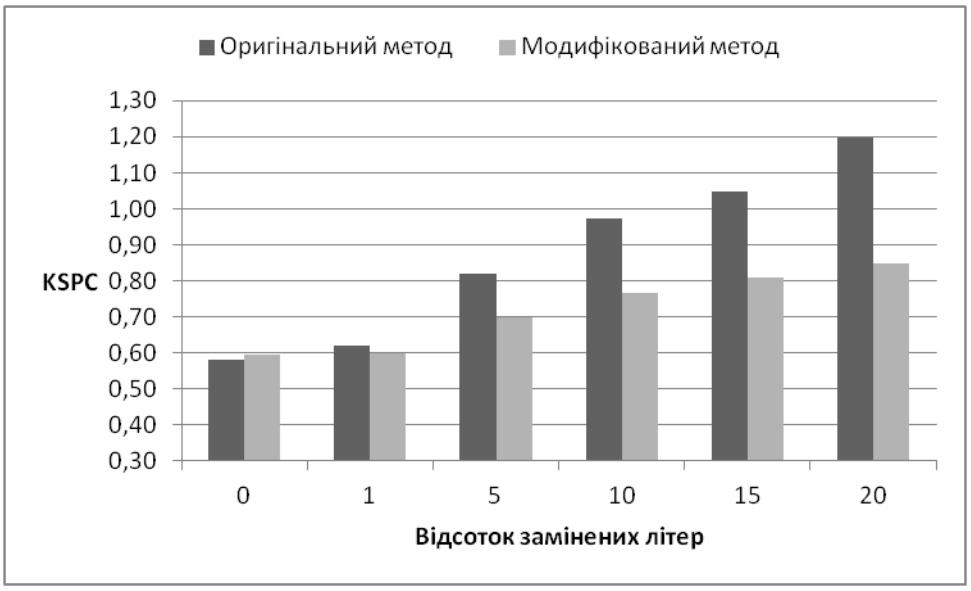


Рис. 29. Порівняння результатів роботи в середньому для всіх текстових корпусів і тексту із загальною лексикою

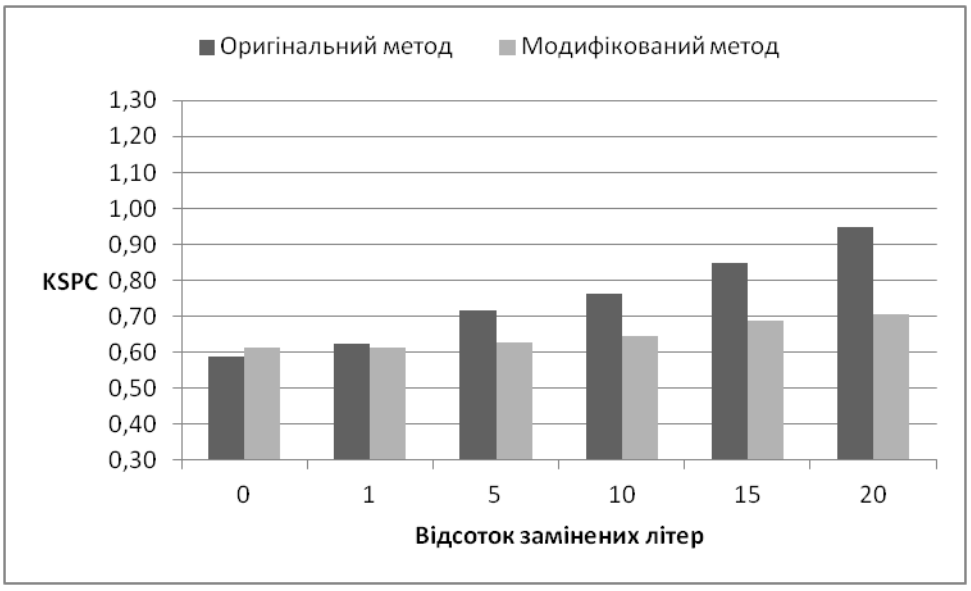


Рис. 30. Порівняння результатів роботи в середньому для всіх текстових корпусів і тексту із розмовною лексикою

Також, слід зазначити, що різниця між значенням KSPC для оригінального та модифікованого методу в середньому є вищою для тексту із загальною лексикою. Це пов’язано з тим, що у тексті із загальною лексикою середня довжина слова є більшою. Відповідно, чим більшою є довжина слова, тим більшу кількість натискань дозволяє зекономити вибір бажаного слова зі списку на початку його введення. Тому у випадках, коли при використанні модифікованого методу бажане слово потрапляє до списку пропозицій для введення, а при використанні оригінального – не потрапляє, більша довжина слова може сприяти збільшенню різниці між значеннями KSPC для цих методів.

На рис. 31 наведено діаграму порівняння результатів роботи оригінального та модифікованого методу в середньому для всіх текстових корпусів та обох текстів для введення.

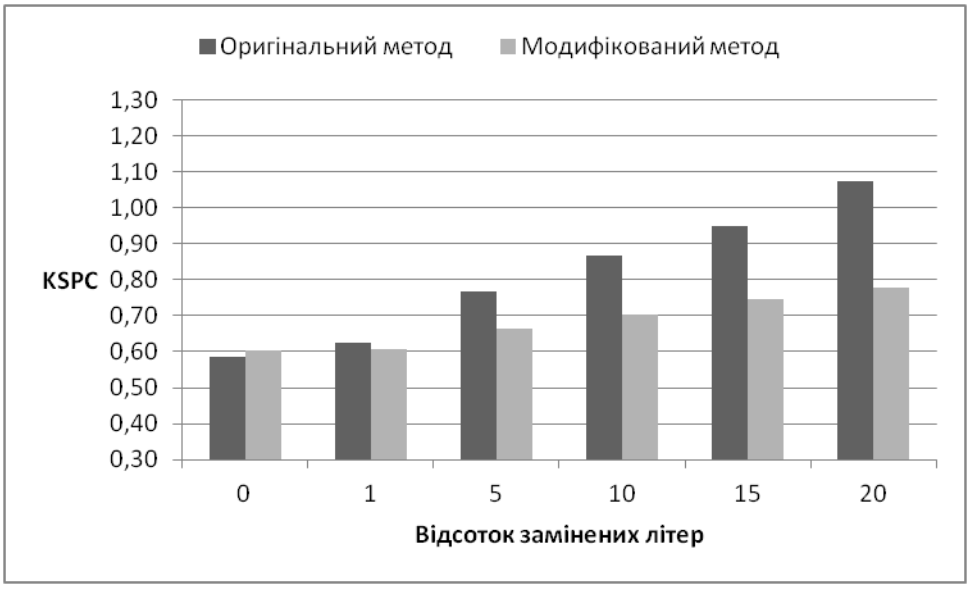


Рис. 31. Порівняння результатів роботи в середньому для всіх текстових корпусів і обох текстів для введення

Отже, в середньому для всіх текстових корпусів і тексту для введення із загальною лексикою значення KSPC для модифікованого методу є нижчим на 14,96%, для тексту для введення із розмовною

лексикою – на 11,86%, а середньому для обох текстів для введення, відповідно, – на 13,55%.

4.3. Шляхи подальшого вдосконалення

Запропонована модифікація методу предиктивного введення тексту на основі *N*-грам враховує можливі помилкові натискання сусідніх літер під час введення тексту користувачем. Однак, у випадку, якщо користувач помилково здійснив натискання деякої літери декілька разів, або ж взагалі помилково не натиснув її, продовживши вводити наступні літери, коректна робота методу не забезпечується, оскільки введена послідовність літер не відповідатиме бажаній, навіть за умови врахування помилкових натискань сусідніх літер. Тому, одним з можливих шляхів подальшого вдосконалення є забезпечення можливості врахування випадків, коли у введеній послідовності літер відсутні деякі літери, або, навпаки, присутні зайві.

Також, слід зазначити, що у запропонованій модифікації методу предиктивного введення тексту на основі *N*-грам при генеруванні списку пропозицій для введення враховуються лише дані про частотний розподіл *N*-грам слів, отримані як в результаті оброблення деякого текстового корпусу, так і в процесі введення тексту користувачем. Однак, при цьому не враховуються дані про частотний розподіл *N*-грам літер, які є достатньо усталеними для тієї чи іншої мови, і, відповідно, не потребують додаткового оброблення текстових корпусів чи оновлення в процесі введення тексту користувачем. Використання даних про частотний розподіл *N*-грам літер може певним чином підвищити ефективність роботи методу у випадку відсутності або недостатньої кількості даних про частотний розподіл *N*-грам слів, наприклад, за відсутності необхідних текстових корпусів для деякої мови. Тому забезпечення можливості врахування даних про частотний розподіл *N*-грам літер також може бути одним з можливих шляхів вдосконалення.

Нарешті, результати проведеного тестування показали, що у випадку відсутності замінених літер оригінальний метод предиктивного введення тексту на основі *N*-грам має незначну перевагу перед запропонованою модифікацією, що становить близько 2-3%. Хоча й така перевага є достатньо малою, одним з можливих шляхів подальшого дослідження може також бути забезпечення ефективності запропонованої модифікації на рівні, не нижчому за рівень ефективності оригінального методу, у випадку відсутності замінених літер.

ВИСНОВКИ

Метою даної магістерської дисертації було підвищення швидкості введення тексту на сучасних мобільних пристроях в умовах обмежених розмірів екранної клавіатури за рахунок розроблення та реалізації модифікованого методу предиктивного введення текстових даних на мобільних пристроях та відповідних програмних засобів.

В ході дослідження, проведеного в рамках даної магістерської дисертації, проаналізовано існуючі методи введення тексту на мобільних пристроях та програмні засоби, що реалізують ці методи. Також охарактеризовано проблему недостатньої ефективності існуючих засобів предиктивного введення тексту на мобільних пристроях.

Для підвищення швидкості введення тексту на мобільних пристроях запропоновано модифікацію методу предиктивного введення тексту на основі N-грам, яка полягає у застосуванні розподілу літер між блоками таким чином, що кожна літера утворює свій окремий блок. Це, в свою чергу, дозволяє забезпечити врахування можливих помилкових натискань сусідніх літер під час швидкого введення тексту, і таким чином, збільшити швидкість введення тексту в умовах обмежених розмірів екранної клавіатури на сучасних мобільних пристроях.

З метою реалізації та тестування запропонованої модифікації методу предиктивного введення тексту на основі N-грам розроблено програмні засоби, що дозволяють користувачеві вводити текст, надаючи йому при цьому пропозиції для введення. Слід зазначити, що розроблені програмні засоби мають можливість пристосування до особливостей введення тексту користувачем, враховуючи вже введені ним слова під час створення наступних пропозицій для введення.

Для дослідження ефективності запропонованої модифікації у порівнянні з оригінальним методом проведено тестування розроблених програмних засобів з використанням даних п’яти текстових корпусів, а

також двох текстів для введення, що переважно містять відповідно загальну та розмовну лексику. Для кожного з двох текстів додатково створено набір з п’яти текстів, що містять відповідно 1%, 5%, 10%, 15% та 20% літер, замінених на сусідні, з метою тестування обох методів в умовах помилкових натискань сусідніх клавіш. У якості критерію ефективності обрано критерій KSPC, що показує кількість натискань, необхідних для введення одного символу.

Результати, отримані в ході тестування, показали, що запропонована модифікація є в середньому на 14,96% ефективнішою за оригінальний метод для тексту із загальною лексикою, та на 11,86% – для тексту із розмовною.

Можливими шляхами подальшого дослідження є забезпечення можливості врахування випадків, коли у введеній послідовності літер відсутні деякі літери, або, навпаки, присутні зайві, забезпечення можливості використання даних про частотний розподіл N-грам літер, а також оптимізація роботи методу за умови відсутності помилкових натискань сусідніх літер.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Desktop vs Mobile Market Share Worldwide [Електронний ресурс]. – Режим доступу: http://gs.statcounter.com/platform-market-share/desktop-mobile/worldwide/#yearly-2009-2019. – Дата доступу: квітень 2019. – Назва з екрана.
2. An Analysis of Power Consumption in a Smartphone [Електронний ресурс]. – Режим доступу: https://www.usenix.org/legacy/event/atc10/tech/full_papers/Carroll.pdf. – Дата доступу: квітень 2019. – Назва з екрана.
3. Hasselgren, J. HMS: A Predictive Text Entry Method Using Bigrams [Text] / J. Hasselgren, E. Montnemery, P. Nugues, M. Svensson // Proceedings of the 2003 EACL Workshop on Language Modeling for Text Entry Methods. – 2003. – P. 43-49.
4. MessagEase [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/MessagEase. – Дата доступу: квітень 2019. – Назва з екрана.
5. Distribution of Word Lengths in Various Languages [Електронний ресурс]. – Режим доступу: http://www.ravi.io/language-word-lengths. – Дата доступу: квітень 2019. – Назва з екрана.
6. What are the longest and shortest languages in terms of average length of words? [Електронний ресурс]. – Режим доступу: https://www.quora.com/What-are-the-longest-and-shortest-languages-in-terms-of-average-length-of-words. – Дата доступу: квітень 2019. – Назва з екрана.
7. English Letter Frequency Counts: Mayzner Revisited or ETAOIN SRHLDCU [Електронний ресурс]. – Режим доступу: http://norvig.com/mayzner.html. – Дата доступу: квітень 2019. – Назва з екрана.

8. Statistics - Sentence and Word Length [Електронний ресурс]. – Режим доступу: http://ds.nahoo.net/Academic/Maths/Sentence.html. – Дата доступу: квітень 2019. – Назва з екрана.
9. Average English word length - Wolfram|Alpha [Електронний ресурс]. – Режим доступу: https://www.wolframalpha.com/input/?i=Average+English+word+length. – Дата доступу: квітень 2019. – Назва з екрана.
10. What is the average length of words in the English language and how does it compare to other languages? [Електронний ресурс]. – Режим доступу: https://www.quora.com/What-is-the-average-length-of-words-in-the-English-language-and-how-does-it-compare-to-other-languages. – Дата доступу: квітень 2019. – Назва з екрана.
11. Кашлер, Клифф [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%88%D0%BB%D0%B5%D1%80,_%D0%9A%D0%BB%D0%B8%D1%84%D1%84. – Дата доступу: квітень 2019. – Назва з екрана.
12. T9 [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/T9. – Дата доступу: квітень 2019. – Назва з екрана.
13. Корпус текстов [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D1%80%D0%BF%D1%83%D1%81_%D1%82%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2. – Дата доступу: квітень 2019. – Назва з екрана.
14. Речевой корпус [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%87%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BA%D0%BE%D1%80%D0%BF%D1%83%D1%81. – Дата доступу: квітень 2019. – Назва з екрана.
15. *N*-грамма [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/N-

%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0. – Дата доступу: квітень 2019. – Назва з екрана.

16. Колокація (лінгвістика) [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D0%BE%D0%BA%D0%B0%D1%86%D1%96%D1%8F_(%D0%BB%D1%96%D0%BD%D0%B3%D0%B2%D1%96%D1%81%D1%82%D0%B8%D0%BA%D0%B0). – Дата доступу: квітень 2019. – Назва з екрана.
17. *N*-gram [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/N-gram. – Дата доступу: квітень 2019. – Назва з екрана.
18. T9 (predictive text) [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/T9_(predictive_text). – Дата доступу: квітень 2019. – Назва з екрана.
19. iTap (predictive text) [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/ITap. – Дата доступу: квітень 2019. – Назва з екрана.
20. Google [Електронний ресурс]. – Режим доступу: https://www.google.com.ua/. – Дата доступу: квітень 2019. – Назва з екрана.
21. A Note on Calculating Text Entry Speed [Електронний ресурс]. – Режим доступу: https://www.yorku.ca/mack/RN-TextEntrySpeed.html. – Дата доступу: квітень 2019. – Назва з екрана.
22. Typing Test @ AOEU — Your typing speed in CPM and WPM [Електронний ресурс]. – Режим доступу: https://typing-speed-test.aoeu.eu/. – Дата доступу: квітень 2019. – Назва з екрана.
23. Метод максимальної правдоподібності [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%BC%D0%B0%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%97_%D0%BF%D1

%80%D0%B0%D0%B2%D0%B4%D0%BE%D0%BF%D0%BE%D0%B4%D1%96%D0%B1%D0%BD%D0%BE%D1%81%D1%82%D1%96. – Дата доступу: квітень 2019. – Назва з екрана.

24. Kryvonos Iu. G. Predictive text typing system for the Ukrainian language [Text] / Iu. G. Kryvonos, Iu. V. Krak, O. V. Barmak, R. O. Bagriy // Cybernetics and Systems Analysis. – 2017. – Vol. 53, № 4. – P. 495-502.
25. Python [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Python. – Дата доступу: квітень 2019. – Назва з екрана.
26. Java [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Java. – Дата доступу: квітень 2019. – Назва з екрана.
27. Perl [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Perl. – Дата доступу: квітень 2019. – Назва з екрана.
28. Python и другие языки программирования: сходство и отличия [Електронний ресурс]. – Режим доступу: https://techrocks.ru/2017/09/18/python-and-other-programming-languages-similarities-and-differences/. – Дата доступу: квітень 2019. – Назва з екрана.
29. Full-text data from English-Corpora.org (COCA, COHA, GloWbE, NOW, Wikipedia, Spanish) [Електронний ресурс]. – Режим доступу: https://www.corpusdata.org/formats.asp. – Дата доступу: травень 2019. – Назва з екрана.

ДОДАТКИ

Додаток 1

Копія презентації

МЕТОД ПРЕДИКТИВНОГО ВВЕДЕННЯ ТЕКСТУ НА МОБІЛЬНИХ ПРИСТРОЯХ

Студент: Копач Сергій Миколайович
Науковий керівник: к.т.н., доцент Заболотня Тетяна Миколаївна

Вступ

- Обмін текстовими повідомленнями користувачами мобільних пристроїв сьогодні набув широкого розповсюдження.
- З огляду на це, а також на обмеженість розмірів екранної клавіатури ефективність засобів введення тексту на мобільних пристроях має велике значення.
- Існує багато методів введення тексту з клавіатури, які дозволяють у тій чи іншій мірі прискорити цей процес.
- Однак більшість з них не можуть бути застосовані до сучасних мобільних пристроїв, оскільки були розроблені ще за часів використання 12-кнопкових клавіатур.
- Новітні методи також не є оптимальними, тому що не враховують можливості помилкового натискання сусідніх літер.

Мета роботи

- Підвищення швидкості введення тексту на сучасних мобільних пристроях в умовах обмежених розмірів екранної клавіатури за рахунок розроблення та реалізації модифікованого методу предиктивного введення текстових даних на мобільних пристроях та відповідних програмних засобів.

Об’єкт та предмет дослідження

- Об’єктом дослідження є процес введення текстових даних на сучасних мобільних пристроях.
- Предметом дослідження є методи, способи та алгоритми предиктивного введення тексту на сучасних мобільних пристроях.

Задачі

- Аналіз існуючих методів введення тексту на мобільних пристроях та програмних засобів, що реалізують ці методи
- Дослідження ефективності засобів введення тексту на мобільних пристроях
- Модифікація методу предиктивного введення тексту на мобільних пристроях
- Розроблення програмного забезпечення для дослідження ефективності запропонованої модифікації методу
- Аналіз отриманих результатів

Критерій ефективності методів введення тексту

- KSPC (keystrokes per character) – кількість натискань, необхідних для введення одного символу.

Поняття багатозначності та однозначності методів введення тексту

- Багатозначний метод: послідовність введених символів може відповідати одночасно декільком словам у словнику
- Однозначний метод: послідовність введених символів може відповідати одночасно лише одному слову у словнику

Існуючі методи введення тексту на мобільних пристроях

- Методи багатьох натискань
- Методи одного натискання
- Методи предиктивного введення тексту на основі *N*-грам

Методи багатьох натискань

- Потребують натискання більше однієї клавіші для введення одного символу
- Дозволяють здійснювати однозначне введення тексту
- Приклади: метод Multi-Tap, метод перевизначеної клавіатури

Методи одного натискання

- Направлені на зменшення кількості натискань, необхідних для введення одного символу, до одиниці
- Дозволяють здійснювати багатозначне введення тексту
- Приклади: метод предиктивного введення тексту, метод WordWise, метод LetterWise

Методи предиктивного введення тексту на основі *N*-грам

- Дозволяють здійснювати прогнозування наступного слова на основі попередніх слів
- Це досягається за рахунок використання даних про ймовірності появи одних слів після інших

N-грами – 1/2

- *N*-грама – деяка послідовність, що складається з *N* елементів (зазвичай літер або слів).

Кількість елементів	Назва *N*-грами
N = 1	уніграма
N = 2	біграма
N = 3	триграма

N-грами – 2/2

- Метою створення уніграм є визначення ймовірностей появи того чи іншого елемента в певному наборі таких елементів.
- Метою створення інших *N*-грам є визначення ймовірностей появи одного елемента після іншого в певному наборі таких елементів.

Приклад процесу створення *N*-грам

	Уніграми	**Біграми**	**Триграми**
Текстовий корпус →	“she” “you” “do” “to” “he” “wants” “are” …	“she” + “wants” “you” + “are” “do” + “you” “would” + “you” “give” + “me” “come” + “on” …	“she” + “wants” + “to” “you” + “are” + “so” “do” + “you” + “have” “would” + “you” + “like” “give” + “me” + “a” …

Визначення ймовірності появи N-грами

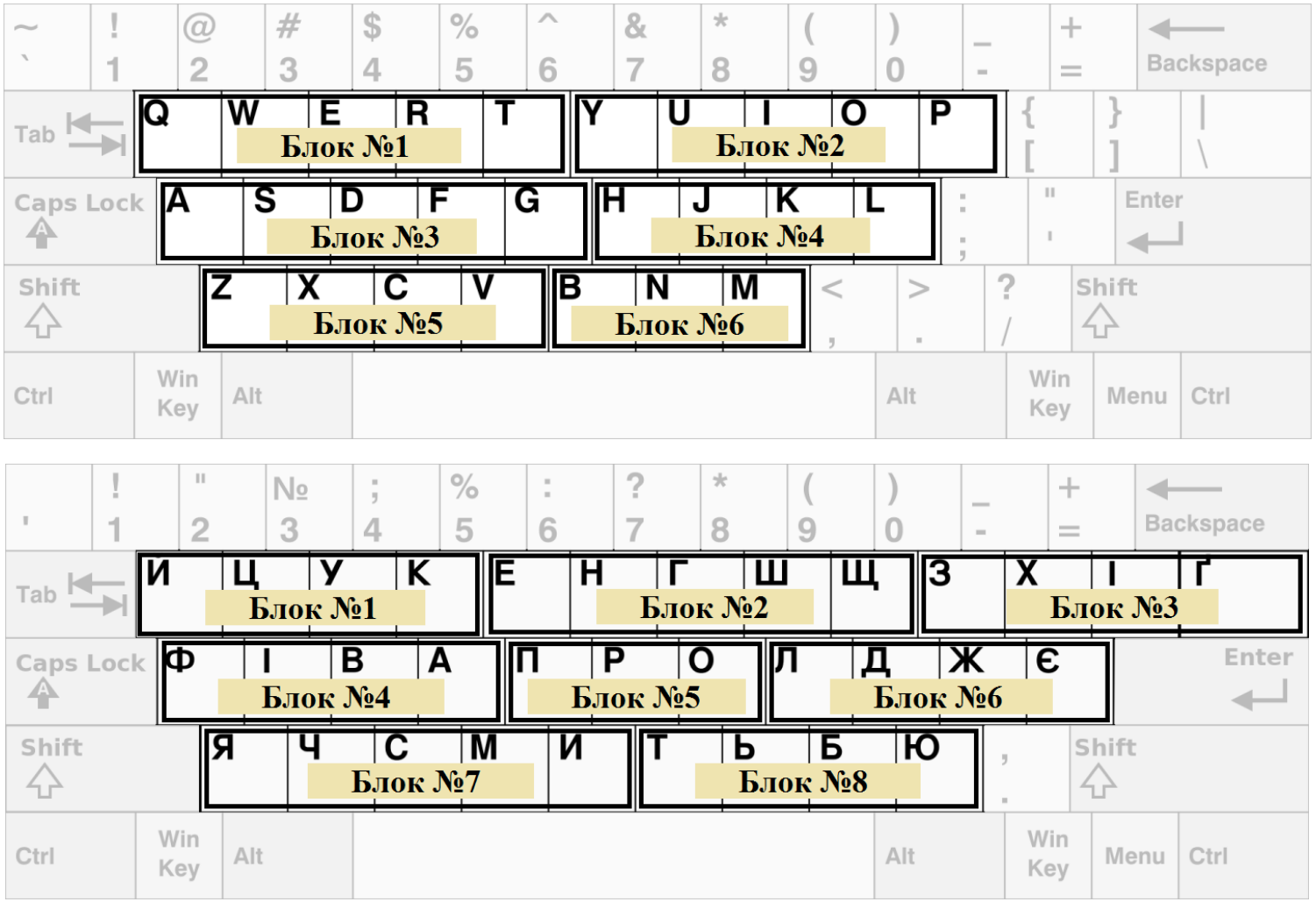
- Ймовірність появи деякої N-грами згідно методу максимальної правдоподібності можна обчислити наступним чином:

$$P\left(w_n \middle| w_{n-N+1}^{n-1}\right) = \frac{C\left(w_{n-N+1}^{n-1}, w_n\right)}{C\left(w_{n-N+1}^{n-1}\right)}$$

Розподіл літер між блоками, що не перетинаються – 1/2

- Використання розподілу літер між блоками таким чином, щоб кожній літері відповідав один блок
- При натисканні літери відбувається вибір усіх літер з блоку, до якого належить ця літера

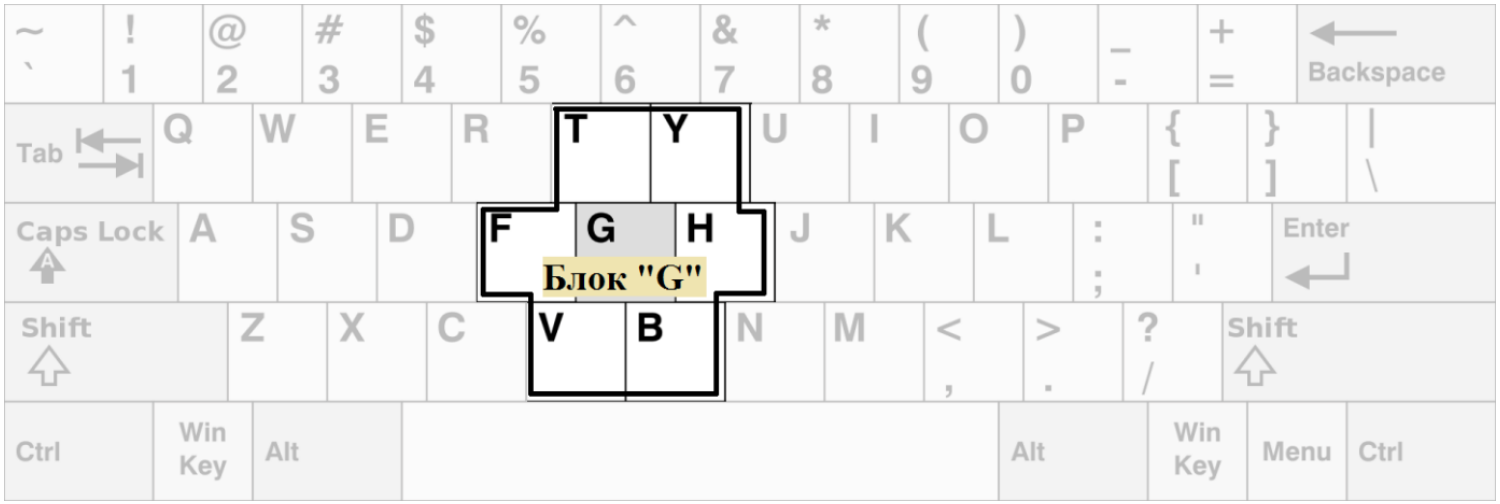
Розподіл літер між блоками, що не перетинаються – 2/2



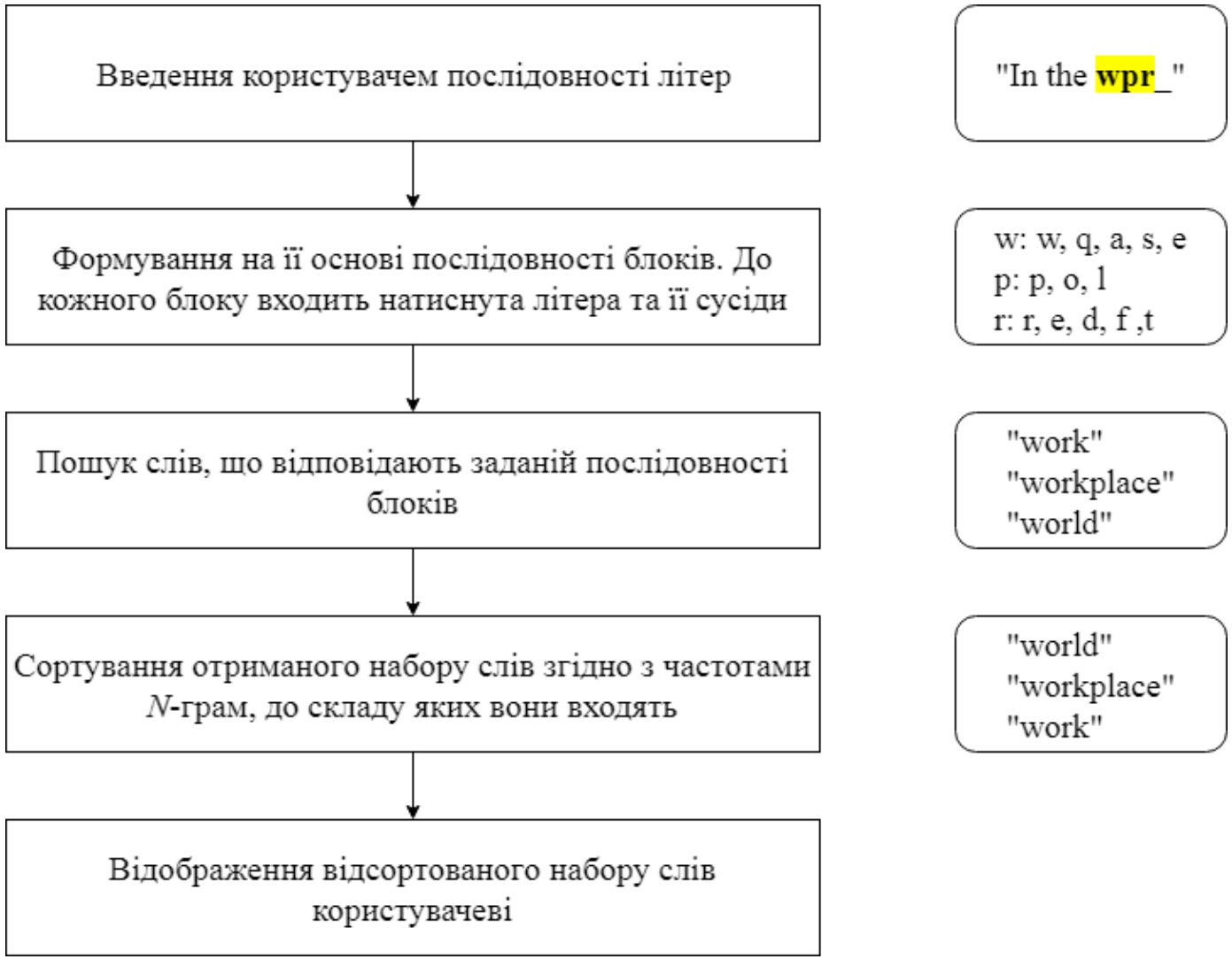
Розподіл літер між блоками, що перетинаються – 1/2

- Використання розподілу літер між блоками таким чином, щоб кожна літера утворювала свій окремий блок
- При натисканні літери відбувається вибір усіх літер з блоку, який утворює ця літера

Розподіл літер між блоками, що перетинаються – 2/2



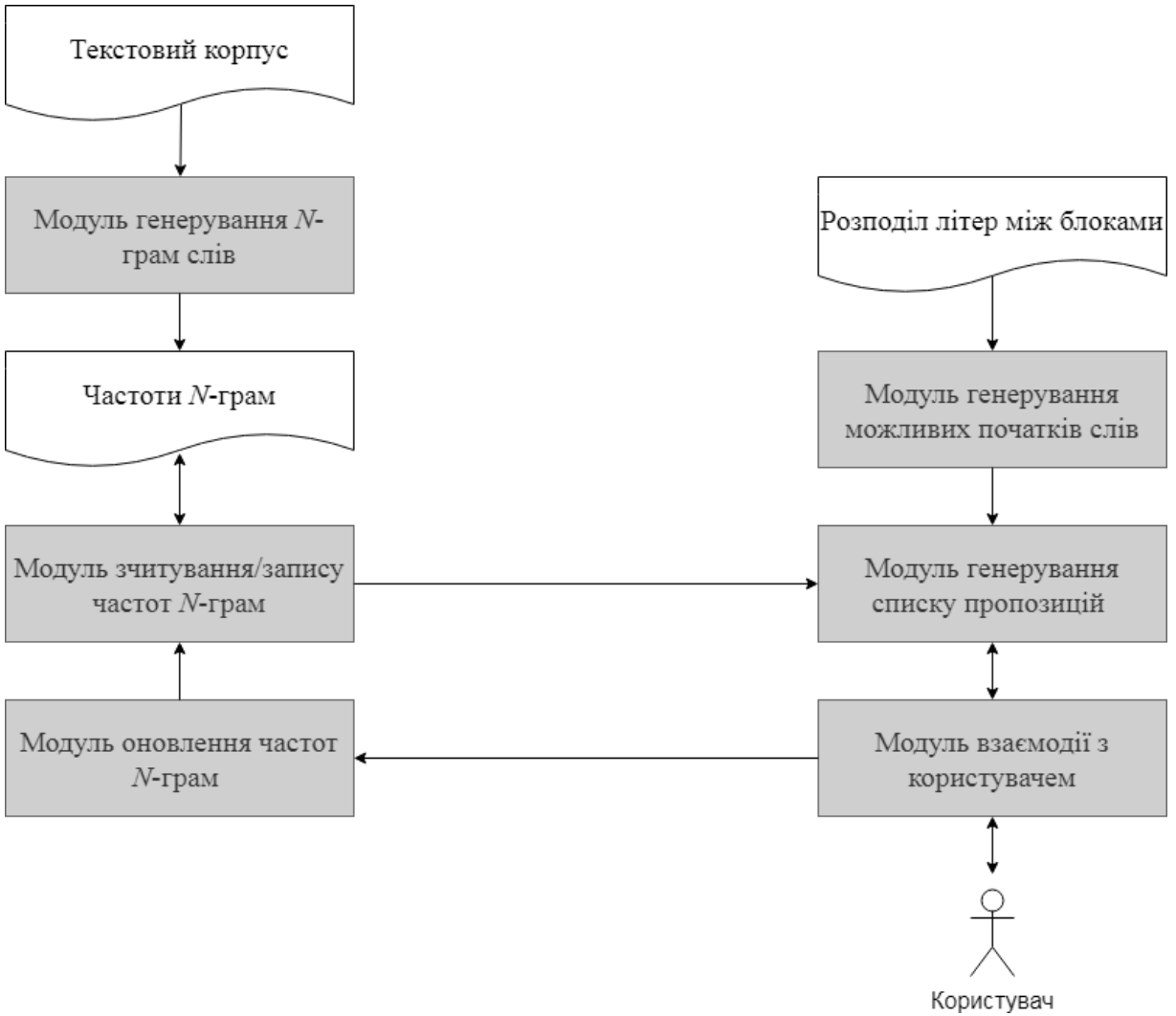
Модифікація методу предиктивного введення тексту на основі *N*-грам



Функціональність розроблених програмних засобів

- Генерування *N*-грам слів та підрахунок їх частот на основі текстового корпусу
- Можливість введення тексту користувачем
- Відображення списку пропозицій для введення користувачеві
- Врахування слів, що вводить користувач, при створенні наступних списків пропозицій для введення

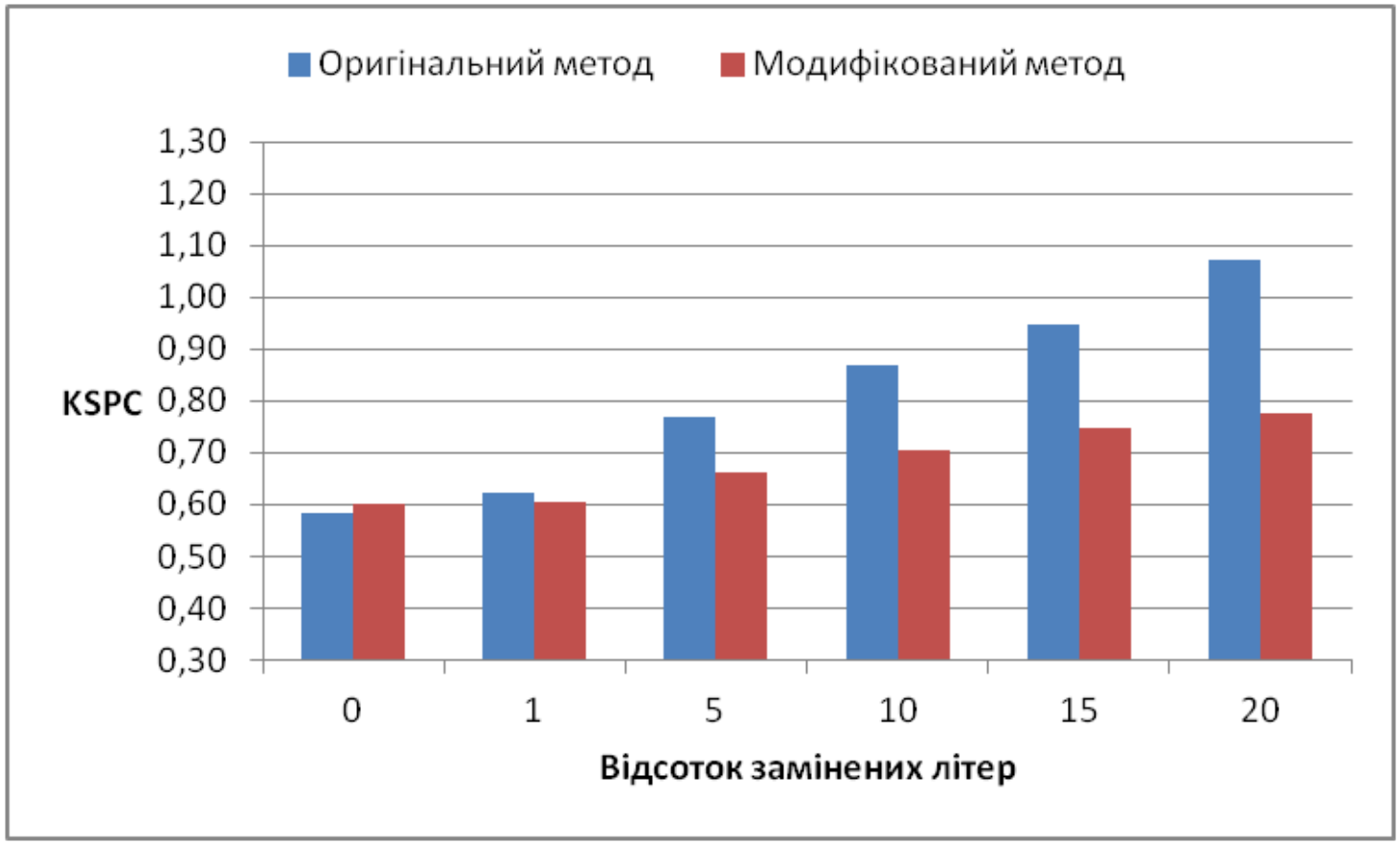
Структура розроблених програмних засобів



Використані технології

- Мова програмування: Python 2.7.11
- Середовище розробки: PyCharm Community Edition 3.4.3
- Операційна система: Windows
- Стандартна бібліотека Python

Результати проведеного тестування



Наукова новизна

- Запропоновано модифікований метод предиктивного введення текстових даних на мобільних пристроях на основі *N*-грам, який відрізняється від існуючих методів застосуванням розподілу літер між блоками таким чином, що кожна літера утворює свій окремий блок, і у такий спосіб сприяє додаванню у список пропозицій необхідних слів навіть при помилкових натисканнях сусідніх літер під час введення тексту, що сприяє прискоренню процесу введення тексту в середньому на 13,41%.

Список публікацій

- Тези доповіді “Модифікований метод предиктивного введення тексту на мобільних пристроях на основі *N*-грам” на XI науковій конференції магістрантів та аспірантів «Прикладна математика та комп’ютинг» ПМК-2018-2 (Київ, 14-16 листопада 2018 р.)
- Стаття “Особливості програмної реалізації модифікованого методу предиктивного введення тексту на мобільних пристроях на основі *N*-грам”, подана до публікації у фаховому виданні “Вісник ЖДТУ. Серія “Технічні науки”

Шляхи подальшого дослідження

- Забезпечення можливості врахування випадків, коли у введеній послідовності літер відсутні деякі літери, або, навпаки, присутні зайві
- Забезпечення можливості використання даних про частотний розподіл *N*-грам літер
- Оптимізація роботи методу за умови відсутності помилкових натискань сусідніх літер

Висновки

- В ході даної роботи було розглянуто існуючі методи введення тексту на мобільних пристроях та програмні засоби, що реалізують ці методи
- Також було запропоновано модифікацію методу предиктивного введення тексту на основі *N*-грам, яка заснована на використанні розподілу літер між блоками таким чином, що кожна літера утворює свій окремий блок.
- Для реалізації та тестування запропонованої модифікації розроблено програмні засоби, що дозволяють вводити текст на отримувати пропозиції для введення.
- Для дослідження ефективності запропонованої модифікації у порівнянні з оригінальним методом проведено тестування розроблених програмних засобів з використанням різних наборів даних.
- Результатом запропонованої модифікації є можливість врахування помилкових натискань сусідніх літер під час введення тексту, що дозволяє підвищити швидкість введення тексту на мобільних пристроях в середньому на 13,41%.

Дякую за увагу!

Додаток 2

Текст програми

Лістинг 1. Тексту модуля n_grams_generator.py, що реалізує генерування *N*-грам слів та підрахунку їх частот на основі деякого текстового корпусу

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import os, re, time
import n_grams_manager

# 0) Obtaining the main directory path

default_main_directory_path = "texts"

raw_main_directory_path = raw_input("Please enter the directory path
(default: {0}): ".format(default_main_directory_path))

main_directory_path = raw_main_directory_path if
len(raw_main_directory_path) > 0 else default_main_directory_path

time_start = time.time()

# 1) Tokenization

print "Tokenization (1/3)..."

tokens = []

for root, subdirs, files in os.walk(main_directory_path):
        for filename in files:
                file_path = os.path.join(root, filename)

                with open(file_path, "r") as f:
                        content = f.readlines()

                        for line in content:
                                parts = line.split(" ")

                                for part in parts:
                                        spaced_part = re.sub(r"[^a-zA-
Z']+", ' ', part)
                                        unspaced_part = re.sub(r" ", '',
spaced_part)
                                        part_tokens = spaced_part.split("
") if len(unspaced_part) > 0 else ['']
                                        tokens += part_tokens

                        tokens.append('')

words = filter(None, tokens)

frequencies = {}

for word in words:
        frequencies[word] = frequencies.get(word, 0) + 1

filtered_frequencies = {}

for key, value in frequencies.iteritems():
        lowercase_key = key.lower()
        filtered_key = ''
```

```
        if frequencies.get(lowercase_key, 0) == 0:
                filtered_key = key
        else:
                difference = float(value) /
float(frequencies.get(lowercase_key, 0))
                filtered_key = key if (difference > 5.0) else lowercase_key

        filtered_frequencies[filtered_key] =
filtered_frequencies.get(filtered_key, 0) + value

formatted_tokens = []

for token in tokens:
        formatted_token = token if ((len(token) == 0) or
(filtered_frequencies.get(token, 0) != 0)) else token.lower()
        formatted_tokens.append(formatted_token)

# 2) Generating the N-grams

print "Generating the N-grams (2/3)..."

total_n = 3

total_n_grams_frequencies = {1: filtered_frequencies}

for n in xrange(2, total_n + 1):
        n_grams_frequencies = {}

        formatted_tokens_count = len(formatted_tokens)

        for i in xrange(n - 1, formatted_tokens_count):
                n_gram = formatted_tokens[(i - n + 1):(i + 1)]

                if len(filter(None, n_gram)) == n:
                        n_gram_tuple = tuple(n_gram)
                        n_grams_frequencies[n_gram_tuple] =
n_grams_frequencies.get(n_gram_tuple, 0) + 1

        total_n_grams_frequencies[n] = n_grams_frequencies

# 3) Writing the N-grams frequencies to the files

print "Writing the N-grams frequencies to the files (3/3)..."

n_grams_manager.save_n_grams_frequencies(total_n_grams_frequencies,
for_user=False)

print "Completed!"

time_end = time.time()
print "Elapsed time: {0} seconds".format(round(time_end - time_start, 3))
```

Лістинг 2. Тексту модуля specific_propositions_creator.py, що реалізує генерування списку пропозицій для введення

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import operator
import candidates_creator

propositions_limit = 3


# Formatting the tokens


def create_formatted_token(token, unigram_frequencies):
        value = unigram_frequencies.get(token, 0)
        if value != 0:
                return token

        value = unigram_frequencies.get(token.lower(), 0)
        if value != 0:
                return token.lower()

        value = unigram_frequencies.get(token.capitalize(), 0)
        if value != 0:
                return token.capitalize()

        value = unigram_frequencies.get(token.upper(), 0)
        if value != 0:
                return token.upper()

        return None


def create_formatted_tokens(tokens, frequencies_general, frequencies_user):
        formatted_tokens = []

        unigram_frequencies_user = frequencies_user.get(1, {})
        unigram_frequencies_general = frequencies_general.get(1, {})

        for token in tokens:
                formatted_token = create_formatted_token(token,
unigram_frequencies_user)
                formatted_token = formatted_token if formatted_token else
create_formatted_token(token, unigram_frequencies_general)
                formatted_token = formatted_token if formatted_token else
token

                formatted_tokens.append(formatted_token)

        return formatted_tokens


# Creating the matches


def create_matches(n_gram, frequencies):
        n = len(n_gram) + 1

        n_grams_frequencies = frequencies.get(n, {})
```

```
        matches = [item for item in n_grams_frequencies.items() if
item[0][:len(n_gram)] == tuple(n_gram)]

        sorted_matches = sorted(matches, key=operator.itemgetter(1),
reverse=True)

        stripped_sorted_matches = sorted_matches[:propositions_limit] if
len(sorted_matches) > propositions_limit else sorted_matches

        results = [x[0][-1:][0] for x in stripped_sorted_matches]

        return results


# Creating candidates matches


def create_possible_beginnings(candidate, unigram_frequencies):
        matches = []

        for key in unigram_frequencies.keys():
                if key.startswith(candidate):
                        matches.append(key)

                elif key.startswith(candidate.lower()):
                        matches.append(key)

                elif key.startswith(candidate.capitalize()):
                        matches.append(key)

                elif key.startswith(candidate.upper()):
                        matches.append(key)

        return matches


def create_beginnings_frequencies(beginnings, n_gram, frequencies):
        beginnings_frequencies = {}

        for beginning in beginnings:
                possible_n_gram = n_gram + [beginning]

                initial_n = len(possible_n_gram)
                for n in xrange(initial_n, 0, -1):
                        current_tokens = possible_n_gram[-n:]

                        key = tuple(current_tokens) if n > 1 else
current_tokens[0]
                        value = frequencies.get(n, {}).get(key, 0)

                        if value > 0:
                                current_frequencies =
beginnings_frequencies.get(n, {})
                                current_frequencies[key] = value
                                beginnings_frequencies[n] =
current_frequencies

        return beginnings_frequencies


def create_beginnings_propositions(beginnings_frequencies, n_gram):
        propositions = []

        initial_n = len(n_gram) + 1
```

```
        for n in xrange(initial_n, 0, -1):
                current_frequencies = beginnings_frequencies.get(n, {})

                sorted_frequencies = sorted(current_frequencies.items(),
key=operator.itemgetter(1), reverse=True)

                if n > 1:
                        propositions += [x[0][-1] for x in
sorted_frequencies if x[0][-1] not in propositions]
                else:
                        propositions += [x[0] for x in sorted_frequencies
if x[0] not in propositions]

        return propositions


def create_candidates_matches(n_gram, candidates, frequencies):
        unigram_frequencies = frequencies.get(1, {})

        first_candidate, other_candidates = candidates[0], candidates[1:]

        first_beginnings = create_possible_beginnings(first_candidate,
unigram_frequencies)
        other_beginnings = []

        if len(first_beginnings) < propositions_limit:
                for other_candidate in other_candidates:
                        other_beginnings +=
create_possible_beginnings(other_candidate, unigram_frequencies)

        if len(n_gram) == 0:
                first_beginnings_frequencies, other_beginnings_frequencies
= {}, {}

                for beginning in first_beginnings:
                        first_beginnings_frequencies[beginning] =
unigram_frequencies.get(beginning, 0)

                for beginning in other_beginnings:
                        other_beginnings_frequencies[beginning] =
unigram_frequencies.get(beginning, 0)

                s_first_beginnings_frequencies =
sorted(first_beginnings_frequencies.items(), key=operator.itemgetter(1),
reverse=True)
                s_other_beginnings_frequencies =
sorted(other_beginnings_frequencies.items(), key=operator.itemgetter(1),
reverse=True)

                s_beginnings_frequencies = s_first_beginnings_frequencies +
s_other_beginnings_frequencies

                propositions_count = min(len(s_beginnings_frequencies),
propositions_limit)
                stripped_s_beginnings_frequencies =
s_beginnings_frequencies[:propositions_count]

                propositions = [x[0] for x in
stripped_s_beginnings_frequencies]

                return propositions

        else:
```

```
                first_beginnings_frequencies =
create_beginnings_frequencies(first_beginnings, n_gram, frequencies)

                other_beginnings_frequencies =
create_beginnings_frequencies(other_beginnings, n_gram, frequencies)

                propositions =
create_beginnings_propositions(first_beginnings_frequencies, n_gram)

                if len(propositions) < propositions_limit:
                        propositions +=
create_beginnings_propositions(other_beginnings_frequencies, n_gram)

                propositions_count = min(len(propositions),
propositions_limit)
                stripped_propositions = propositions[:propositions_count]

                return stripped_propositions


# Creating propositions for an N-gram


def create_propositions_n_gram(tokens, frequencies_general,
frequencies_user):
        finish = False

        formatted_tokens = create_formatted_tokens(tokens,
frequencies_general, frequencies_user)

        current_tokens = formatted_tokens
        matches = []

        while not finish:
                current_matches = create_matches(current_tokens,
frequencies_user)
                matches += [x for x in current_matches if x not in matches]

                if len(matches) < propositions_limit:
                        current_matches = create_matches(current_tokens,
frequencies_general)
                        matches += [x for x in current_matches if x not in
matches]

                if (len(matches) < propositions_limit) and
len(current_tokens) > 1:
                        current_tokens = current_tokens[1:]

                else:
                        finish = True

        matches_count = min(len(matches), propositions_limit)
        stripped_matches = matches[:matches_count]

        return stripped_matches


# Creating propositions for a unigram


def create_propositions_unigram(tokens, frequencies_general,
frequencies_user):
```

```
        candidates =
candidates_creator.create_candidates(tokens[len(tokens) - 1],
frequencies_general, frequencies_user)

        if len(candidates) > 0:
                formatted_tokens = create_formatted_tokens(tokens,
frequencies_general, frequencies_user)

                n_gram = formatted_tokens[:-1]

                matches = create_candidates_matches(n_gram, candidates,
frequencies_user)

                if len(matches) < propositions_limit:
                        matches += create_candidates_matches(n_gram,
candidates, frequencies_general)

                last_token = tokens[len(tokens) - 1]

                format_match = lambda x: x.capitalize() if x.islower() else
x

                formatted_matches = matches if last_token.islower() else
[format_match(x) for x in matches]

                filtered_matches = []

                for x in formatted_matches:
                        if x not in filtered_matches:
                                filtered_matches.append(x)

                stripped_filtered_matches_count =
min(len(filtered_matches), propositions_limit)
                stripped_filtered_matches =
filtered_matches[:stripped_filtered_matches_count]

                return stripped_filtered_matches

        else:
                return []


# Creating propositions for a new unigram


def create_propositions_new_unigram(frequencies_general, frequencies_user):
        unigrams_frequencies_general = frequencies_general.get(1)
        unigrams_frequencies_user = frequencies_user.get(1)

        sorted_unigrams_frequencies = []

        if unigrams_frequencies_user:
                sorted_unigrams_frequencies +=
sorted(unigrams_frequencies_user.items(), key=operator.itemgetter(1),
reverse=True)

        if (len(sorted_unigrams_frequencies) < propositions_limit) and
unigrams_frequencies_general:
                sorted_unigrams_frequencies +=
sorted(unigrams_frequencies_general.items(), key=operator.itemgetter(1),
reverse=True)

        propositions = [x[0] for x in sorted_unigrams_frequencies]
```

```
        filtered_propositions = []

        for x in propositions:
                if x not in filtered_propositions:
                        filtered_propositions.append(x)

        stripped_filtered_propositions_count =
min(len(filtered_propositions), propositions_limit)
        stripped_filtered_propositions =
filtered_propositions[:stripped_filtered_propositions_count]

        return stripped_filtered_propositions


# Creating empty propositions


def create_propositions_nothing():
        return []
```

Лістинг 3. Тексту модуля candidates_creator.py, що реалізує генерування можливих початків слів в залежності від розподілу літер між блоками

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

letter_blocks_filename = "letter_blocks.txt"


def create_letter_blocks():
        with open(letter_blocks_filename, "r") as f:
                content = f.readlines()
                blocks_list = [line.rstrip() for line in content]

                blocks_dict = {}
                for block in blocks_list:
                        blocks_dict[block[0]] = tuple(block)
                        #blocks_dict[block[0]] = tuple(block[0])

                return blocks_dict


def create_words(frequencies_general, frequencies_user):
        unigrams_frequencies_user = frequencies_user.get(1, {})
        unigrams_frequencies_general = frequencies_general.get(1, {})

        words = list(set(unigrams_frequencies_user.keys() +
unigrams_frequencies_general.keys()))
        return words


blocks = create_letter_blocks()


def create_candidates(token, frequencies_general, frequencies_user):
        words = create_words(frequencies_general, frequencies_user)

        starts_with = lambda x, candidate: (x.startswith(candidate) or
x.startswith(candidate.lower()))

        candidates = []
        current_words = []

        for i in xrange(0, len(token)):
                ch = token[i]

                ch_candidates = blocks.get(ch.lower(), [ch])

                f_ch_candidates = ch_candidates if ch.islower() else
[x.upper() for x in ch_candidates]

                if i == 0:
                        for f_ch_candidate in f_ch_candidates:
                                for word in words:
                                        if starts_with(word,
f_ch_candidate):
                                                if f_ch_candidate not in
candidates:

candidates.append(f_ch_candidate)

                                                if word not in
current_words:
```

```
current_words.append(word)

                else:
                        new_candidates = []
                        new_current_words = []

                        for candidate in candidates:
                                for f_ch_candidate in f_ch_candidates:
                                        new_candidate = candidate +
f_ch_candidate

                                        for word in current_words:
                                                if starts_with(word,
new_candidate):
                                                        if new_candidate
not in new_candidates:

new_candidates.append(new_candidate)

                                                        if word not in
new_current_words:

new_current_words.append(word)

                        candidates = new_candidates
                        current_words = new_current_words

        return candidates
```